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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

Тема: Стек

Студент гр. 7383 _____ Зуев Д.В.

Преподаватель _____ Размочева Н.В.

Санкт-Петербург

2018

ОГЛАВЛЕНИЕ

Цель работы.....3

Реализация задачи.....3

Тестирование.....5

Вывод.....5

Приложение А. Тестовые случаи.....6

Приложение Б. Исходный код программы.....7

Цель работы

Цель работы: познакомиться со структурой и реализацией стека, получить навыки использования их в практических задачах на языке программирования C++.

Формулировка задачи: Вариант 6-в. 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*

Реализация задачи

В данной работе используются класс Stack — стек на базе массива. Переменные класса Stack:

- char* stackPtr — массив символов, в котором хранятся элементы стека.
- int size — размер массива.
- int top — номер верхнего элемента стека.

Методы класса Stack:

- Stack(int = 10) — конструктор, создающий стек с максимальным количеством элементов, равным 10.
- ~Stack — деструктор, очищает массив элементов.
- void newStack(int) - увеличивает максимально-возможное количество помещаемых в стек элементов путем создания заново массива элементов и удвоения текущего размера массива.
- int getTop() - возвращает номер верхнего элемента.
- void push(const char) - добавляет в элемент в стек. Если номер последнего элемента равен размеру массива элементов, то вызывает функцию newStack.
- char pop() - достает и возвращает верхний элемент из стека.
- char tops() - возвращает верхний элемент стека.

В функции main выводится приглашение выбрать способ ввода входных данных либо выйти из программы. В случае выбора файла, программа считывает текст из файла и записывает его в поток ввода. В случае ввода информации с консоли функция main считывает строку с консоли, записывает эту строку в этот же поток ввода.

Функция readFormula получает на вход поток входных данных.
Бросает ошибку если строка пустая. Считывает строку, пропуская пробелы , если встретит неприемлемые символы, бросает ошибку. Выводит считанную строку на экран и возвращает её.

Функция checkFormula проходит по каждому элементу полученной на вход строки. Если элемент закрывающая скобка, тогда вызывает функцию checkBrackets, иначе кладет элемент в стек. Если после прохождения всех элементов в стеке не один элемент бросает ошибку.

Функция checkBrackets проверяет выражение внутри скобок. Если относительно закрывающей скобки на месте имени или операции стоит не то, что нужно, бросает ошибку. Если на месте закрывающей скобки стоит закрывающая скобка другого, бросает ошибку.

Функции isClosingBrackets и isName ппроверяют символ на то, является ли он закрывающей функцией или именем соответственно.

Так же были реализованы классы ошибок. Каждый класс — под отдельную ошибку.

Тестирование

Программа была собрана в компиляторе G++ в среде разработки Qt creator в OS Linux Ubuntu 16.04 LTS.

В ходе тестирования была найдена ошибка. При первом вызове функции checkFormula элементы не записывались в стек. Причиной ошибки было отсутствие присвоения переменной size класса Stack какого-либо значения в конструкторе. Ошибка была исправлена.

Корректные тестовые случаи представлены в приложении А.

Вывод

В ходе работы были получены навыки работы со стеком на базе массива на языке C++. Поскольку строку со скобочной записью проверить пройдя один раз от начала до конца затруднительно, для проверки строки удобно использовать стек.

ПРИЛОЖЕНИЕ А.

ТЕСТОВЫЕ СЛУЧАИ

Таблица 1 — Корректные случаи

Входные данные	Выходные данные
d	Wrong character. The character "d" is not int.
12	You entered wrong number.
1 (d+x)	The following character is incorrect. Incorrect character is "d" Formula is wrong.
1 ++x	The following character is incorrect. Incorrect character is "+"
1 xyx	The input string: x y x The following character must be an operation. The character "y" must be an operation. Formula is wrong.
1 (x+{z+y])	The input string: (x + { z + y]) The opening bracket is not confirm to the closing bracket. The bracket "{" does not match the bracket "]". Formula is wrong.
1 (x+y	(x + y Missing closing bracket. Formula is wrong.
1 x+z)	The input string: x + z) Missing opening bracket. Formula is wrong.
2	The input string: x + y + z Formula is right.
1 z-(x+{x+y}+z)	The input string: z - (x + { x + y } + z) Formula is right.
1 [x+y]-{y-z}	The input string: [x + y] - { y - z } Formula is right.

ПРИЛОЖЕНИЕ Б.

ИСХОДНЫЙ КОД ПРОГРАММЫ

main.cpp:

```
#include <iostream>
#include <sstream>
#include <fstream>
#include <cstring>
#include <exception>
#include <cstdlib>
#include "functions.hpp"
using namespace std;

int main()
{
    stringstream xstream;
    char* str0;
    str0 = new char[100];
    string tmp1;
    short int tmp = 0;
    while(tmp != 3)
    {
        try{
            xstream.str("");
            xstream.clear();
            cout<<"Введите 1, если желаете вводить выражение с
клавиатуры.\n"
              "Введите 2, если желаете брать выражение из файла
test.txt.\n"
              "Введите, 3 если хотите закончить работу."<<endl;
            getline(cin, tmp1);
            if(!atoi(tmp1.c_str() ))
                throw error9(tmp1.c_str());
            else tmp = atoi(tmp1.c_str());
            switch(tmp){
                case 1:
                    cout << "Введите формулу: \n";
                    cin.getline(str0, 100);
                    xstream << str0;
                    break;
                case 2:
                {
                    ifstream outfile;
                    outfile.open("test.txt");
                    if (!outfile)
                        throw error10();
                    outfile.read(str0, 100);
                    outfile.close();
                    xstream << str0;
                    break;
                }
```

```
            case 3:
                continue;
            default:
                throw error11();
        }
        char formula[100];
        strcpy(formula, readFormula(xstream));
        checkFormula(formula);
    }
    catch(error1& e)
    {
        cout<<e.what();
        cout<<"Formula is wrong.\n";
        continue;
    }
    catch(error2& e)
    {
        cout<<e.what();
        cout<<"Formula is wrong.\n";
        continue;
    }
    catch(error3& e)
    {
        cout<<e.what();
        cout<<"Formula is wrong.\n";
        continue;
    }
    catch(error4& e)
    {
        cout<<e.what();
        e.printErr();
        cout<<"Formula is wrong.\n";
        continue;
    }
    catch(error5& e)
    {
        cout<<e.what();
        e.printErr();
        cout<<"Formula is wrong.\n";
        continue;
    }
    catch(error6& e)
    {
        cout<<e.what();
        e.printErr();
        cout<<"Formula is wrong.\n";
        continue;
    }
    catch(error7& e)
    {
        cout<<e.what();
        e.printErr();
        cout<<"Formula is wrong.\n";
        continue;
```

```
        }
        catch(error8& e)
        {
            cout<<e.what();
            cout<<"Formula is wrong.\n";
            continue;
        }
        catch(error9& e)
        {
            cout<<e.what();
            e.printErr();
            continue;
        }
        catch(error10& e)
        {
            cout<<e.what();
            continue;
        }
        catch(error11& e)
        {
            cout<<e.what();
            continue;
        }
        catch(exception& e)
        {
            cout<<e.what();
            continue;
        }
        cout<<"Formula is right.\n";
    }
    return 0;
}
```

functions.cpp:

```
#include <iostream>
#include <sstream>
#include <fstream>
#include <cstring>
#include <exception>
#include "functions.hpp"

Stack::Stack(int maxSize)
{
    stackPtr = new char[maxSize];
    top = 0;
    size = maxSize;
}
Stack::~Stack()
{
    delete [] stackPtr;
}
void Stack::newStack(int newSize)
```

```
{
    char* newPtr = new char[newSize];
    for(int i = 0; i<=size;i++)
        newPtr[i] = stackPtr[i];
    delete [] stackPtr;
    stackPtr = newPtr;
    size = newSize;
}
int Stack::getTop()
{
    return top;
}
void Stack::push(const char value)
{
    if(top < size)
        stackPtr[++top] = value;
    else
    {
        newStack(size<<1);
        stackPtr[++top] = value;
    }
}
char Stack::pop()
{
    if(top != 0)
        return stackPtr[--top];
    else throw error1();
}
char Stack::tops()
{
    if(top != 0)
        return stackPtr[top];
    else throw error2();
}
bool isClosingBrackets(char x)
{
    return x==')'||x=='}'||x==']';
}
bool isName(char x)
{
    return x=='x'||x=='y'||x=='z';
}
char* readFormula(stringstream& xstream)
{
    char x;
    int i = 0;
    char a[100];
    a[i++]='(';
    if(!(xstream>>x))
        throw error3();
    while (x==' ')
        xstream >> x;
    if(x =='('||x=='['||x =='{'||isName(x))
    {
```

```
        a[i++] = x;
    }
    else throw error4(x);
    while(xstream>>x)
    {
        while(x == ' ')
            xstream>>x;
        if(x == '('||x == '['||x =='{'||isName(x)||
isClosingBrackets(x)||x=='+'||x=='-')
        {
            a[i++] = x;
        }
        else throw error4(x);
    }
    a[i++]=')';
    a[i++]='\0';
    cout<<"The input string:\n";
    for(int j = 1;j<i-2;j++)
        cout<<a[j]<<' ';
    cout<<endl;
    return a;
}
void checkFormula(char* a)
{
    Stack s;
    int len = strlen(a);
    for(int i = 0; i < len; i++)
    {
        if(isClosingBrackets(a[i]))
            checkBrackets(a[i], s);
        else
            s.push(a[i]);
    }
    if(s.getTop()!=1 )
        throw error8();
}
void checkBrackets(char br, Stack &s)
{
    char op1, op2, op3;
    switch(br){
    case ')':
        op1 = '(';
        op2 = '{';
        op3 = '[';
        break;
    case '}':
        op1 = '{';
        op2 = '(';
        op3 = '[';
        break;
    case ']':
        op1 = '[';
        op2 = '(';
        op3 = '{';
```

```
            break;
        }
        while(true)
        {
            if(!isName(s.tops()))
            {
                cout<<s.tops()<<endl;
                throw error5(s.tops());
            }
            s.pop();
            if(s.tops() == op2||s.tops() == op3)
                throw error7(s.tops(), br);
            if(s.tops() == op1)
            {
                s.pop();
                s.push('x');
                break;
            }
            if(!(s.tops()=='-'||s.tops()=='+'))
                throw error6(s.tops());
            s.pop();
            if(!isName(s.tops()))
                throw error5(s.tops());
            s.pop();
            if(s.tops() == op2||s.tops() == op3)
                throw error7(s.tops(), br);
            if(s.tops() == op1)
            {
                s.pop();
                s.push('x');
                break;
            }
            s.push('x');
        }
        s.pop();
        s.push('x');
}
```

functions.hpp:

```
#include <iostream>
#include <sstream>
#include <fstream>
#include <exception>

using namespace std;

class Stack
{
private:
    char *stackPtr;
    int size;
    int top;
```

```
public:
    Stack(int = 10);
    ~Stack();

    void newStack(int );
    int getTop();
    void push(const char );
    char pop();
    char tops();
};

class error1: public exception
{
public:
    explicit error1()
        { }

    virtual const char* what() const throw()
    { return "The stack is emptied during the execution of the
function.\n"; }
};
class error2: public exception
{
public:
    explicit error2()
        { }

    virtual const char* what() const throw()
    { return "Missing opening bracket.\n"; }
};
class error3: public exception
{
public:
    explicit error3()
        { }

    virtual const char* what() const throw()
    { return "The input stream is empty.\n"; }
};
class error4: public exception
{
public:
    explicit error4(char a)
        {this->a = a; }

    virtual const char* what() const throw()
    { return "The following character is incorrect.\n"; }
    void printErr()
    {
        cout<<"Incorrect character is \""<<a<<'\"'<<endl;
    }
private:
    char a;
};
```

```
class error5: public exception
{
public:
    explicit error5(char a)
        { this->a = a; }

    virtual const char* what() const throw()
    { return "The following character must be a name.\n"; }
    void printErr()
    {
        cout<<"The character \""<<a<<"\" must be a name."<<endl;
    }
private:
    char a;
};
class error6: public exception
{
public:
    explicit error6(char a)
        {this->a = a; }

    virtual const char* what() const throw()
    { return "The following character must be an
operation.\n"; }
    void printErr()
    {
        cout<<"The character \""<<a<<"\" must be an
operation."<<endl;
    }
private:
    char a;
};
class error7: public exception
{
public:
    explicit error7(char b1, char b2)
        {this->b1=b1; this->b2=b2; }

    virtual const char* what() const throw()
    { return "The opening bracket is not confirm to the closing
bracket.\n"; }
    void printErr()
    {
        cout<<"The bracket \""<<b1<<"\" does not match the
bracket \""<<b2<<"\"."<<endl;
    }
private:
    char b1;
    char b2;
};
class error8: public exception
{
public:
    explicit error8()
```

```
        { }

    virtual const char* what() const throw()
    { return "Missing closing bracket.\n"; }
};
class error9: public exception
{
public:
    explicit error9(const char* a)
        {this->a=a; }

    virtual const char* what() const throw()
    { return "Wrong character.\n"; }
    void printErr()
    {
        cout<<"The character \""<<a<<"\" is not int.\n";
    }
private:
    const char* a;
};
class error10: public exception
{
public:
    explicit error10()
        { }

    virtual const char* what() const throw()
    { return "The input file is not opened.\n"; }
};
class error11: public exception
{
public:
    explicit error11()
        { }

    virtual const char* what() const throw()
    { return "You entered wrong number.\n"; }
};
inline bool isClosingBrackets(char );
inline bool isName(char );
char* readFormula(stringstream& );
void checkFormula(char * );
void checkBrackets(char , Stack&);
```