Министерство образования и науки РФ
Санкт-Петербургский политехнический университет Петра Великого Институт компьютерных наук и технологий
Высшая школа программной инженерии

ОТЧЕТ

по дисциплине «Алгоритмы и структуры данных»

Реализация Абстрактный Тип Данных Stack

Работу выполнил:
студент группы 3530904/00006
Смирнов Е.А.

Работу принял
преподаватель Павлов Е. А.

Санкт-Петербург
2021

Постановка задачи:

1. На основе шаблона для стека создать шаблон для реализации структуры данных «ограниченный» стек (через массив) – **StackArray**

2. Создать классы для работы с двумя основными исключительными ситуациями, которые могут возникнуть при работе со стеком.
   - **StackOverflow** [переполнение]
   - **StackUnderflow** [недостаточно]
   - **WrongStackSize** [для работы с исключительной ситуацией, которая может возникнуть, если в конструкторе стека, реализуемого через массив, неправильно задан размер]

3. Используя шаблон «ограниченный» стек написать
   - функцию анализа правильности расстановки скобок ***checkBalanceBrackets***
   - функцию перевода арифметического выражения из инфиксной формы в постфиксную ***getPostfixFromInfix***
   - функцию вычисления значения арифметического выражения в постфиксной форме ***evaluatePostfix*** для *полной скобочной формы*

4. Создана функция для тестирования всех методов и функций ***Test***

Реализация:

Stack.h

```
#ifndef STACK_HPP
#define STACK_HPP

template <class T>
class Stack
{
public:
  virtual ~Stack() = default;
  virtual void push(const T& e) = 0; // добавление элемента в стек
  virtual const T& pop() = 0; // удаление и возвращение верхнего элемента. // если элементов нет, может возникнуть
StackUnderflow
  virtual bool isEmpty() const = 0; // проверка стека на пустоту
};

#endif
```

StackArray.h

```
#ifndef STACK_ARRAY_HPP
#define STACK_ARRAY_HPP

#include "Stack.h"

template <class T>
class StackArray : public Stack<T> {

public:
  StackArray(int size = 30); // size задает размер стека "по умолчанию"

  StackArray(const StackArray<T>& src);
  StackArray(StackArray<T>&& src);
  StackArray& operator=(const StackArray<T>& src);
  StackArray& operator=(StackArray<T>&& src);
  ~StackArray() override;

  void push(const T& e);
  const T& pop();
  const size_t& getTop();
  bool isEmpty() const noexcept;

private:
  T* array_; // массив элементов стека: !!! array_[0] – не используется, top_ от 1 до size_
  size_t top_; // вершина стека, элемент занесенный в стек последним
  size_t size_; // размер стека
  void swap(StackArray<T>& src) noexcept;
};

#endif
```

StackArray.cpp

```
#include"StackArray.h"

#include <utility>
#include "StackOverflow.h"
#include "StackUnderflow.h"
#include "WrongStackSize.h"

// метод swap - закрытый
template<class T>
inline void StackArray<T>::swap(StackArray<T>& src) noexcept
{
  std::swap(array_, src.array_);
  std::swap(top_, src.top_);
  std::swap(size_, src.size_);
}

// конструктор
template<class T>
StackArray<T>::StackArray(int size) :
        size_(size),
        top_(0)
{
        try { // массив будем начнать с 1, в [0] - ничего нет
                array_ = new T[size + 1]; // заказ памяти под stackarray
        }
        catch (...) { // ... - если что-то случилось
                throw WrongStackSize();
        }
}

// конструктор копирования
template<class T>
```

```
StackArray<T>::StackArray(const StackArray<T>& src) :
	size_(src.size_),
	top_(src.top_)
{
	try
	{
		array_ = new T[src.size_ + 1];
	}
	catch (...)
	{
		throw WrongStackSize();
	}

	// копирование массива
	for (int i = 0; i <= src.top_; i++) {
		array_[i] = src.array_[i];
	}
}

// конструктор перемещения
template<class T>
StackArray<T>::StackArray(StackArray<T>&& src)
	//size_(src.size_),
	//top_(src.top_),
	//array_(src.array_)
{
	swap(src);
	src.size_ = 0;
	src.top_ = 0;
	src.array_ = nullptr;
}

// оператор копирования
template<class T>
StackArray<T>& StackArray<T>::operator=(const StackArray<T>& src)
{
	// очищаем все слева
	delete[] array_;
	array_ = nullptr;

	// копируем из r-v в l-v
	size_ = src.size_;
	top_ = src.top_;

	try
	{
		array_ = new T[src.size_ + 1];
	}
	catch (...)
	{
		throw WrongStackSize();
	}
	for (int i = 0; i <= src.top_; i++) {
		array_[i] = src.array_[i];
	}
}

// оператор перемещения
template<class T>
StackArray<T>& StackArray<T>::operator=(StackArray<T>&& src)
{
	// очищаем все слева
	delete[] array_;
	array_ = nullptr;

	// перемещаем
	size_ = src.size_;
	top_ = src.top_;
	array_ = src.array_;
```

```
	// очищаем r-v
	src.size_ = 0;
	src.top_ = 0;
	src.array_ = nullptr;
}

// деструткор
template<class T>
StackArray<T>::~StackArray()
{
	delete[] array_;
	array_ = nullptr;
	size_ = 0;
	top_ = 0;
}

template<class T>
void StackArray<T>::push(const T& e)
{
	if (top_ == size_) {
		throw StackOverflow(); // нет места для нового элемента
	}
	else {
		array_[++top_] = e; // занесение элемента в стек
	}
}

template<class T>
const T& StackArray<T>::pop()
{
	// стек - пуст
	if (top_ == 0) {
		throw StackUnderFlow();
	}
	// элемент "физически" остается
	return *(array_+top_--); // адресная
}

template<class T>
const size_t& StackArray<T>::getTop()
{
	return top_;
}

template<class T>
bool StackArray<T>::isEmpty() const noexcept
{
	if (top_<size_)
		return true;
	else return false;
}
```

Классы исключительных ситуаций — наследники от std::exception

StackOverflow.h

```
#ifndef STACK_OVERFLOW_HPP
#define STACK_OVERFLOW_HPP

#include<exception>
#include <string>

class StackOverflow : public std::exception
{
public:
	StackOverflow() : reason_("StackOverflow") {}
	const char* what() const { return reason_; }
private:
	const char* reason_;
};
#endif
```

StackUnderFlow.h

```
#ifndef STACK_UNDERFLOW_HPP
#define STACK_UNDERFLOW_HPP

#include<exception>
#include <string>

class StackUnderFlow : public std::exception
{
public:
	StackUnderFlow() : reason_("StackUnderFlow") {}
	const char* what() const { return reason_; }
private:
	const char* reason_;
};
#endif
```

WrongStackSize.h

```
#ifndef STACK_WRONG_STACKSIZE_HPP
#define STACK_WRONG_STACKSIZE_HPP

#include<exception>
#include <string>

class WrongStackSize : public std::exception
{
public:
	WrongStackSize() : reason_("WrongStackSize") {}
	const char* what() const { return reason_; }
private:
	const char* reason_;
};
#endif
```

checkBalanceBrackets – функция, проверяющая правильную вложенность скобок в выражении

```
bool CheckBalanceBrackets(const char* text, int stackSize) {
  StackArray<char> stack(stackSize);
  bool isTrue = true;
  try {
    for (int i = 0; *(text + i) != '\0' && isTrue == true; i++) {
      switch (*(text + i))
      {
      case '(': stack.push(*(text + i)); break;
      case '[': stack.push(*(text + i)); break;
      case '{': stack.push(*(text + i)); break;
      case ')':
        if (stack.pop() != '(')
        {
          isTrue = false;
        }
        break;
      case ']':
        if (stack.pop() != '[')
        {
          isTrue = false;
        }
        break;
      case '}':
        if (stack.pop() != '{')
        {
          isTrue = false;
        }
      }
    }
  }
  catch (...) {
    return false;
  }
  if (stack.getTop()) {
    return false;
  }
  else return isTrue;
}
```

getPostfixFromInfix – функция, создающая постфиксную форму выражения из инфиксной формы

```
bool GetPostfixFromInfix(const char* infix, char* postfixi, size_t stackSize) {
  StackArray<char> stack(stackSize);
  bool isTrue = true;
  char lastInStack;
  int num = 0;
  char pre_item = ' ';
  int availableSigns = 0;
  int i = 0;

  try {
    if (!CheckBalanceBrackets(infix, stackSize)) {
      throw std::invalid_argument("Ошибка в скобках!");
      return false;
    }
    for (i; *(infix + i) != '\0' && isTrue == true; i++) {
      if (!((*(infix + i) > 41) && (*(infix + i) < 58)) && *(infix + i) != ' ' && *(infix +
i) != '(' && *(infix + i) != ')') {
        throw std::invalid_argument("Incorrect symbols");
      }
      switch (*(infix + i)) {

      case '(':
        availableSigns++;
        stack.push(*(infix + i));
        pre_item = '(';
        break;

      case '0':
        if (pre_item > 47 && pre_item < 58) throw std::invalid_argument("Incorrect number");
        *(postfixi + num) = '0';
        num++;
        pre_item = '0';
        break;

      case '1':
        if (pre_item > 47 && pre_item < 58) throw std::invalid_argument("Incorrect number");
        *(postfixi + num) = '1';
        num++;
        pre_item = '1';
        break;
      case '2':
        if (pre_item > 47 && pre_item < 58) throw std::invalid_argument("Incorrect number");
        *(postfixi + num) = '2';
        num++;
        pre_item = '2';
        break;
      case '3':
        if (pre_item > 47 && pre_item < 58) throw std::invalid_argument("Incorrect number");
        *(postfixi + num) = '3';
        num++;
        pre_item = '3';
        break;
      case '4':
        if (pre_item > 47 && pre_item < 58) throw std::invalid_argument("Incorrect number");
        *(postfixi + num) = '4';
        num++;
        pre_item = '4';
```

```
      break;
    case '5':
      if (pre_item > 47 && pre_item < 58) throw std::invalid_argument("Incorrect number");
      *(postfixi + num) = '5';
      num++;
      pre_item = '5';
      break;
    case '6':
      if (pre_item > 47 && pre_item < 58) throw std::invalid_argument("Incorrect number");
      *(postfixi + num) = '6';
      num++;
      pre_item = '6';
      break;
    case '7':
      if (pre_item > 47 && pre_item < 58) throw std::invalid_argument("Incorrect number");
      *(postfixi + num) = '7';
      num++;
      pre_item = '7';
      break;
    case '8':
      if (pre_item > 47 && pre_item < 58) throw std::invalid_argument("Incorrect number");
      *(postfixi + num) = '8';
      num++;
      pre_item = '8';
      break;
    case '9':
      if (pre_item > 47 && pre_item < 58) throw std::invalid_argument("Incorrect number");
      *(postfixi + num) = '9';
      num++;
      pre_item = '9';
      break;

    case '+':
      if (pre_item > 41 && pre_item < 48) throw std::invalid_argument("Incorrect signs");
      if (availableSigns==0)  throw std::invalid_argument("Не полная скобочная форма");
      pre_item = '+';
      stack.push(*(infix + i));
      availableSigns--;
      break;
    case '-':
      if (pre_item > 41 && pre_item < 48) throw std::invalid_argument("Incorrect signs");
      if (availableSigns == 0)  throw std::invalid_argument("Не полная скобочная форма");
      if (pre_item == '(') {
        *(postfixi + num) = '0';
        num++;
      }
      pre_item = '-';
      stack.push(*(infix + i));
      availableSigns--;
      break;
    case '/':
      if (pre_item > 41 && pre_item < 48) throw std::invalid_argument("Incorrect signs");
      if (availableSigns == 0)  throw std::invalid_argument("Не полная скобочная форма");
      pre_item = '/';
      stack.push(*(infix + i));
      availableSigns--;
      break;
    case '*':
      if (pre_item > 41 && pre_item < 48) throw std::invalid_argument("Incorrect signs");
      if (availableSigns == 0)  throw std::invalid_argument("Не полная скобочная форма");
      pre_item = '*';
```

```
        stack.push(*(infix + i));
        availableSigns--;
        break;
      case ')':
        do {
          lastInStack = stack.pop();
          if (lastInStack != '(') {
            *(postfixi + num) = lastInStack;
            num++;
          }

        } while (lastInStack != '(');

      }
    }
  }
  catch (std::exception& e) {
    std::cerr << e.what()<< std::endl;
    return false;
  }
  if (i == 1) return false;
  else return true;
}
```

evaluatePostfix – функция вычисления значения арифметического выражения в постфиксной форме

```
int EvaluatePostfix(const char* infix, size_t stackSize) {
  StackArray<int> stack(stackSize);
  int tempsub = 0;
  int tempdev = 0;
  char* postfixi = new char[stackSize];
  try {
    if (!(GetPostfixFromInfix(infix, postfixi, stackSize))){
      throw std::invalid_argument("Incorrect Formula");
    }


    for (int i = 0; *(postfixi + i) != '\0'; i++) { // ( + )
      switch (*(postfixi + i)) {

      case '0': stack.push(0); break;
      case '1': stack.push(1); break;
      case '2': stack.push(2); break;
      case '3': stack.push(3); break;
      case '4': stack.push(4); break;
      case '5': stack.push(5); break;
      case '6': stack.push(6); break;
      case '7': stack.push(7); break;
      case '8': stack.push(8); break;
      case '9': stack.push(9); break;

      case '+':
        stack.push(stack.pop() + stack.pop());
        break;
      case '-':
        tempsub = stack.pop();
        stack.push(stack.pop() - tempsub);
        break;
      case '/':
        tempdev = stack.pop();
        stack.push(stack.pop() / tempdev);
        break;
      case '*':
        stack.push(stack.pop() * stack.pop());
        break;
      }
    }
    delete[] postfixi;

  }
  catch (std::exception& e) {
    std::cerr << e.what() << std::endl;
    delete[] postfixi;
    return 3;
  }
  return stack.pop();
}
```

MAIN.CPP

```cpp
#include <iostream>
#include "StackArray.cpp"
bool TestEvaluate();
bool TestSteck();
bool CheckBalanceBrackets(const char* , int stackSize = 30);
bool GetPostfixFromInfix(const char* infix, char* postfix, size_t stackSize = 30);
int EvaluatePostfix(const char* infix, size_t stackSize = 30);
bool TestCheckBalanceBrackets();


int main()
{
  setlocale(LC_ALL, "Rus");
  TestCheckBalanceBrackets();
  TestSteck();
  TestEvaluate();
  return 0;
}

bool TestCheckBalanceBrackets()
{
  std::cout << std::endl<< "Test TestCheckBalanceBrackets" << std::endl<< std::endl;
  const char* text00 = " ok ";
  std::cout << text00 << ": " << (CheckBalanceBrackets(text00) ? "right" : "wrong") <<
std::endl;
  const char* text01 = "( ) ok ";
  std::cout << text01 << ": " << (CheckBalanceBrackets(text01) ? "right" : "wrong") <<
std::endl;
  const char* text02 = "( ( [] ) ) ok ";
  std::cout << text02 << ": " << (CheckBalanceBrackets(text02) ? "right" : "wrong") <<
std::endl;
  const char* text03 = "( ( [ { } [ ] ( [ ] ) ] ) ) OK";
  std::cout << text03 << ": " << (CheckBalanceBrackets(text03) ? "right" : "wrong") <<
std::endl;
  const char* text04 = "( ( [ { } [ ] ( [ ] ) ] ) ) ) extra right parenthesis ";
  std::cout << text04 << ": " << (CheckBalanceBrackets(text04) ? "right" : "wrong") <<
std::endl;
  const char* text05 = "( ( [{ }[ ]([ ])] ) extra left parenthesis ";
  std::cout << text05 << ": " << (CheckBalanceBrackets(text05) ? "right" : "wrong") <<
std::endl;
  const char* text06 = "( ( [{ ][ ]([ ])]) ) unpaired bracket ";
  std::cout << text06 << ": " << (CheckBalanceBrackets(text06) ? "right" : "wrong") <<
std::endl;
  return true;
}
bool TestEvaluate() {
  try {

    std::cout << std::endl<< "Test TestEvaluate" << std::endl<< std::endl;

    std::cout << "(1+2*5+1-1*9)" << std::endl;
    std::cout << EvaluatePostfix("(1+2*5+1-1*9)") << std::endl;

    std::cout << "(5 * 3 + 3 * 5)" << std::endl;
    std::cout << EvaluatePostfix("(5 * 3 + 3 * 5)") << std::endl;

    std::cout << "((5 * 3) + (3 * 5))" << std::endl;
    std::cout << EvaluatePostfix("((5 * 3) + (3 * 5))") << std::endl;
```

```
    std::cout << "(2+1)" << std::endl;
    std::cout << EvaluatePostfix("(2+1)") << std::endl;

    std::cout << "(3*0)" << std::endl;
    std::cout << EvaluatePostfix("(3*0)") << std::endl;

    std::cout << "(4/2)" << std::endl;
    std::cout << EvaluatePostfix("(4/2)") << std::endl;

    std::cout << "((-1)+9)" << std::endl;
    std::cout << EvaluatePostfix("((-1)+9)") << std::endl;

    std::cout << "((1-2)*4)" << std::endl;
    std::cout << EvaluatePostfix("((1-2)*4)") << std::endl;

    std::cout << "(((1-2))*(-4))" << std::endl;
    std::cout << EvaluatePostfix("(((1-2))*(-4))") << std::endl;

    std::cout << "((((1+3)*(1+3))*((1+3)*(1+3)))*(((1+3)*(1+3))*((1+3)*(1+3))))" <<
std::endl;
    std::cout <<
EvaluatePostfix("((((1+3)*(1+3))*((1+3)*(1+3)))*(((1+3)*(1+3))*((1+3)*(1+3))))", 50) <<
std::endl;

    std::cout << "((11-2)*4)" << std::endl;
    std::cout << EvaluatePostfix("((11-2)*4)") << std::endl;

    std::cout << "(((1a-2))*(-4))" << std::endl;
    std::cout << EvaluatePostfix("(((1a-2))*(-4))") << std::endl;

    std::cout << "((01-2)*4)" << std::endl;
    std::cout << EvaluatePostfix("((01-2)*4)") << std::endl;

    std::cout << "(((1-2))=(-4))" << std::endl;
    std::cout << EvaluatePostfix("(((1-2))=(-4))") << std::endl;

    std::cout << "1" << std::endl;
    std::cout << EvaluatePostfix("1") << std::endl;

    std::cout << "(-1)" << std::endl;
    std::cout << EvaluatePostfix("(-1)") << std::endl;

    std::cout << "( + )" << std::endl;
    std::cout << EvaluatePostfix("( + )") << std::endl;

    std::cout << "(5-2)*3)" << std::endl;
    std::cout << EvaluatePostfix("(5-2)*3)") << std::endl;

    std::cout << "{ 3 + 2 }" << std::endl;
    std::cout << EvaluatePostfix("{ 3 + 2 }") << std::endl;

    std::cout << "[2-2)3)" << std::endl;
    std::cout << EvaluatePostfix("[2-2)3)") << std::endl;

    std::cout << " " << std::endl;
    std::cout << EvaluatePostfix(" ") << std::endl;


  }
  catch (std::exception& e) {
```

```
    std::cerr << e.what();
    return 3;
  }
  return 0;
}
bool TestSteck() {
  std::cout << std::endl<< "Test TestSteck" << std::endl<<std::endl;
  try {
    StackArray<int> stack1(2);
    stack1.push(20);
    stack1.push(10);
    // stack1.push(1); -- overflow
    // stack1.pop();
    // stack1.pop();
    // stack1.pop();  -- undeflow
    // StackArray<int> stack2(-1); // --wrongsize

    StackArray<int> stack3(1); // оператор копирования
    stack3 = stack1;
    std::cout << stack3.pop() << std::endl << stack3.pop() << std::endl;

    StackArray<int> stack4 = std::move(stack1); // конструктор перемещения
    std::cout << stack4.pop() << std::endl << stack4.pop() << std::endl;
  }
  catch (std::exception& e) {
    std::cerr << e.what();
    return 3;
  }
  return 0;
                                                }
```

Выходные значения:

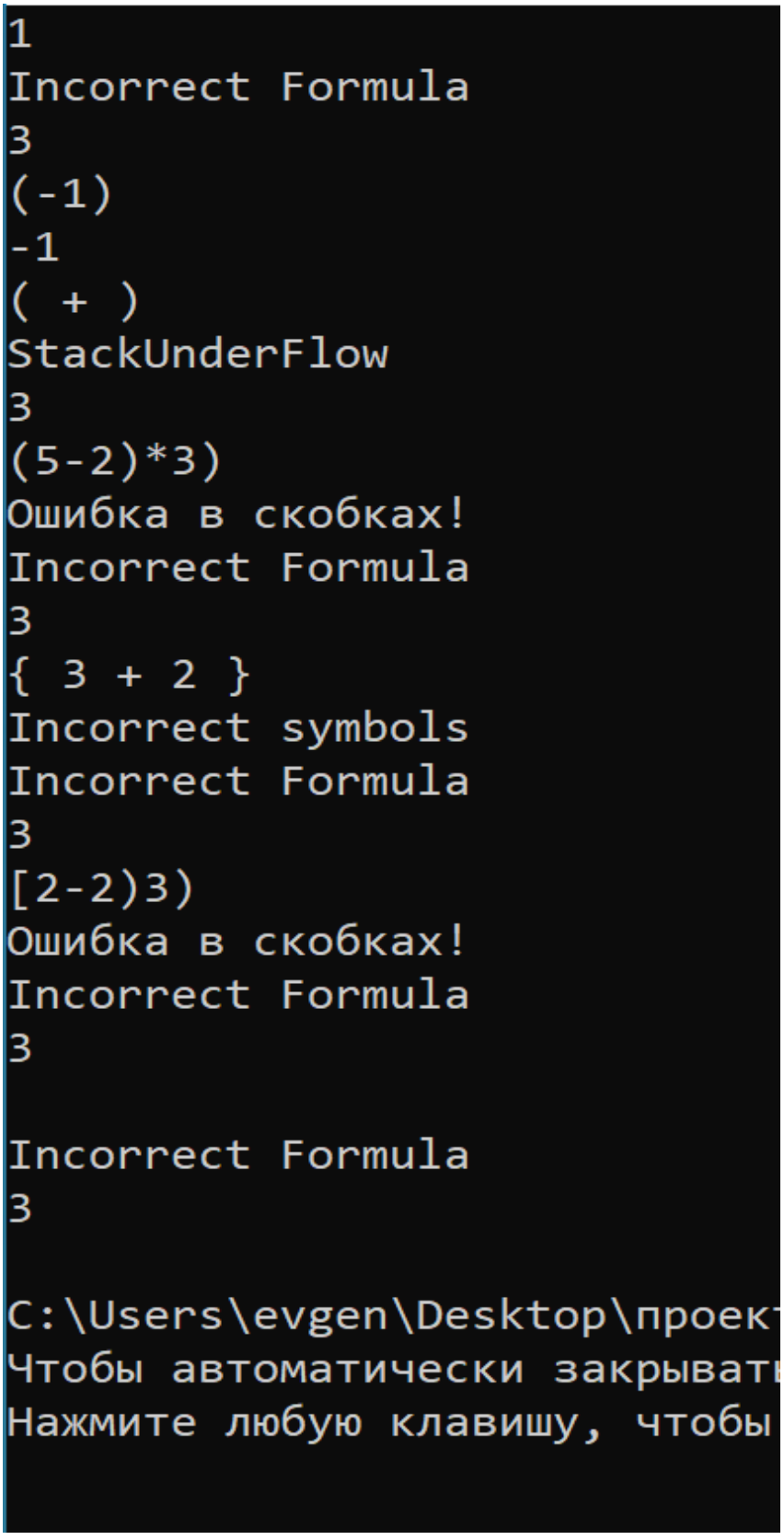
1.

```
Test TestCheckBalanceBrackets

 ok : right
( ) ok : right
( ( [] ) ) ok : right
( ( [ { } [ ] ( [ ] ) ] ) ) OK: right
( ( [ { } [ ] ( [ ] ) ] ) ) ) extra right parenthesis : wrong
( ( [{ }[ ]([ ])] ) extra left parenthesis : wrong
( ( [{ ][ ]([ ])]) ) unpaired bracket : wrong

Test TestSteck

20
10
20
10
```

2.

```
Test TestEvaluate

(1+2*5+1-1*9)
Не полная скобочная форма
Incorrect Formula
3
(5 * 3 + 3 * 5)
Не полная скобочная форма
Incorrect Formula
3
((5 * 3) + (3 * 5))
30
(2+1)
3
(3*0)
0
(4/2)
2
((-1)+9)
8
((1-2)*4)
-4
(((1-2))*(-4))
4
((((1+3)*(1+3))*((1+3)*(1+3)))*(((1+3)*(1+3))*((1+3)*(1+3))))
3
((11-2)*4)
Incorrect number
Incorrect Formula
3
(((1a-2))*(-4))
Incorrect symbols
Incorrect Formula
3
((01-2)*4)
Incorrect number
Incorrect Formula
3
(((1-2))=(-4))
Incorrect symbols
Incorrect Formula
3
```

3.


1
Incorrect Formula
3
(-1)
-1
(+)
StackUnderFlow
3
(5-2)*3)
Ошибка в скобках!
Incorrect Formula
3
{ 3 + 2 }
Incorrect symbols
Incorrect Formula
3
[2-2)3)
Ошибка в скобках!
Incorrect Formula
3
Incorrect Formula
3
C:\Users\evgen\Desktop\проек
Чтобы автоматически закрыват
Нажмите любую клавишу, чтобы