**САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО**

**ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК И КИБЕРБЕЗОПАСНОСТИ**

**ВЫСШАЯ ШКОЛА ПРОГРАММНОЙ ИНЖЕНЕРИИ**

**Лабораторная работа №2**

**«Структуры данных: стек и очередь»**

**по предмету «Алгоритмы и структуры данных»**

**Выполнил: студент гр. 5130904/30002 Севостьянова А.В.**

**Руководитель: Череповский Д.К.**

**Санкт-Петербург**

**2024 г.**

Оглавление

[1. Общая постановка задачи 3](#_Toc160368000)

[1. Реализуйте структуру данных «ограниченный» стек 3](#_Toc160368001)

[2. Реализуйте функцию анализа правильности расстановки скобок 3](#_Toc160368002)

[3. Реализуйте структуру данных «ограниченная» очередь 3](#_Toc160368003)

[2. Детальные требования и тест план 4](#_Toc160368004)

[Детальные требования 4](#_Toc160368005)

[Таблица с детальными требованиями и тест планом 4](#_Toc160368006)

[3. Программа 6](#_Toc160368007)

[main.cpp 6](#_Toc160368008)

[Stack.h 7](#_Toc160368009)

[Queue.h 9](#_Toc160368010)

[Functions.h 11](#_Toc160368011)

[Functions.cpp 11](#_Toc160368012)

[Приложение A 13](#_Toc160368013)

[Приложение B 14](#_Toc160368014)

[Вывод 15](#_Toc160368015)

1. **Общая постановка задачи**

## *1. Реализуйте структуру данных «ограниченный» стек*

1. Используйте шаблон для абстрактного типа данных класса стек:

template <class T>

class Stack

{

public:

virtual ~Stack() {}

virtual void push(const T& e) = 0; // Добавление элемента в стек

virtual T pop() = 0; // Удаление верхнего элемента

virtual bool isEmpty() = 0; // Проверка стека на пустоту

};

2. На основе шаблона для стека создайте шаблон для реализации структуры данных «ограниченный» стек (через массив).

3. Создайте классы StackOverflow и StackUnderflow для работы с двумя исключительными ситуациями, которые могут возникнуть при работе со стеком.

4. Создайте класс WrongStackSize для работы с исключительной ситуацией, которая может возникнуть, если в конструкторе стека, реализуемого через массив, неправильно задан размер.

5. Выполните тестирование созданного класса.

## *2. Реализуйте функцию анализа правильности расстановки скобок*

Функция должна возвращать True, если количество открывающих и закрывающих скобок одного типа совпадает, и они имеют правильную вложенность. Допускаются три вида скобок: круглые, квадратные и фигурные.

Прототип функции:

bool checkBalanceBrackets (const char\* text, const int maxDeep);

или

bool checkBalanceBrackets (const string& text, const int maxDeep);

text - анализируемый текст, содержащий скобки

maxDeep - максимально возможный уровень вложенности скобок

В реализации используйте шаблон «ограниченный стек».

## *3. Реализуйте структуру данных «ограниченная» очередь*

1. Используйте шаблон для абстрактного типа данных класса очередь:

template <class T>

class Queue

{

public:

virtual ~Queue () {}

virtual void enQueue(const T& e) = 0; // Добавление элемента в очередь

virtual T deQueue() = 0; // Удаление элемента из очереди

virtual bool isEmpty() = 0; // Проверка очереди на пустоту

};

2. На основе шаблона для очереди создайте шаблон для реализации структуры данных «ограниченная» кольцевая очередь (через массив).

3. Создайте классы QueueOverflow и QueueUnderflow для работы с двумя исключительными ситуациями, которые могут возникнуть при работе с очередью.

4. Создайте класс WrongQueueSize для работы с исключительной ситуацией, которая может возникнуть, если в конструкторе очереди, реализуемой через массив, неправильно задан размер.

5. Выполните тестирование созданного класса

# 2. Детальные требования и тест план

## *Детальные требования*

1. Размеры стека и очереди должны быть заданы корректно (положительное целое число)

2. Элементы стека и очереди должны быть заданы корректно

3. Размеры стека и очереди не должны превышать максимально возможного.

4. Количество элементов в стеке и очереди не может превышать его размер

5. Удаление элементов из массива и очереди возможно только тогда, когда в них содержаться элементы

## *Таблица с детальными требованиями и тест планом*

|  |  |  |  |
| --- | --- | --- | --- |
| Требования | Детальные требования | Данные | Ожидаемый результат |
| 1. Размеры стека и очереди – целое положительное число | Если размер стека или очереди не является целым положительным числом, то сообщение «Неверный размер стека (очереди)», завершение работы программы. | -7  0  Stst  4.5 | Сообщение:  «Неверный размер стека (очереди)»  Завершение работы программы. |
| 2. Размеры стека и очереди заданы корректно | Если размеры стека и очереди – целое положительное число, то продолжение работы программы | 4 | Продолжение работы программы. |
| 3. Элементы стека и очереди должны быть заданы корректно (тестирование для типа int –> целые числа) | Если элементы не принимают целочисленное значение, то сообщение:  «Некорректный ввод», завершение работы программы. | 4.5  tdtdt | Сообщение:  «Некорректный ввод»  Завершение работы программы. |
| 4.Элементы стека и очереди заданы корректно | Если элементы стека и очереди – целые числа, продолжение работы программы | 4 | Продолжение работы программы. |
| 5. Размеры стека и очереди не должны превышать максимально возможного. | Если размер стека или очереди превышает максимально возможный, то сообщение «Неверный размер стека (очереди)», завершение работы программы. | 100000000 | Сообщение:  «Неверный размер стека (очереди)»  Завершение работы программы. |
| 6. Размеры стека и очереди не превышают максимально возможного. | Если размер стека и очереди не превышают максимально возможный, продолжение работы программы | 5 | Продолжение работы программы. |
| 7. Количество элементов в стеке и очереди не должно превышать его размер. | Если при добавлении нового элемента в стек или очередь происходит переполнение, то сообщение:  «Стек(Очередь) переполнен(а)», заверение работы программы. | Переполнение. | Сообщение:  «Стек(Очередь) переполнен(а)»  Заверение работы программы. |
| 8. Количество элементов в стеке и очереди не превышает из размеров. | Если при добавление нового элемента не происходит переполнение структуры, продолжение работы программы. | Переполнение не происходит. | Продолжение работы программы. |
| 9. Удаление элементов возможно только тогда, когда структура не пуста. | Если стек или очередь не содержат элементов, то удаление невозможно, сообщение:  «Стек (очередь) пуст(а)», завершение работы программы. | Удаление из пустой структуры. | Сообщение:  «Стек (очередь) пуст(а)»  Завершение работы программы. |
| 10. Удаление элементов возможно только тогда, когда структура не пуста | Если при попытки удаление в структуре содержаться элементы. Продолжение работы программы. | Структура не пуста. | Продолжение работы программы. |

# 3. Программа

## *main.cpp*

#include "Stack.h"

#include"Queue.h"

#include"Functions.h"

const std::string ERRROR\_OF\_INPUT = "Ошибка ввода\n";

int main()

{

setlocale(LC\_ALL, "ru");

try

{

{

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Stack(int)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n";

std::cout << "Введите количество элементов стека:\t";

std::string size="";

std::cin >> size;

StackArray<int> stack(size);

int cnt = 0;

std::string value = "";

while (cnt < std::stoi(size))

{

std::cout << "Введите элемент:\t";

std::cin >> value;

checkInput(value) ? stack.push(std::stoi(value)) : throw std::invalid\_argument(ERRROR\_OF\_INPUT);

cnt++;

}

stack.print();

std::cout << "Удаление элемента:\t"<<stack.pop()<<"\n";

stack.print();

stack.isEmpty() ? std::cout << "Stack пуст\n" : std::cout << "В Stack есть элементы\n";

}

{

std::cout << "\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Queue(int)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n";;

std::cout << "Введите количество элементов очереди:\t";

std::string size = "";

std::cin >> size;

CircularQueue<int> queue(size);

int cnt = 0;

std::string value = "";

while (cnt < std::stoi(size))

{

std::cout << "Введите элемент:\t";

std::cin >> value;

checkInput(value) ? queue.enQueue(std::stoi(value)) : throw std::invalid\_argument(ERRROR\_OF\_INPUT);

cnt++;

}

queue.print();

std::cout << "Удаление элемента:\n";

queue.deQueue();

queue.print();

queue.print();

queue.isEmpty() ? std::cout << "Очередь пуст\n" : std::cout << "В очереди есть элементы\n";

}

std::cout << "\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_checkBalanceBrackets\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n";

std::cout << "Введите выражение для проверки соответсвия скобок:\t";

std::string text = "";

std::cin >> text;

std::cout << "Введите максимальную глубину вложенности скобок:\t";

int maxDeep = 0;

std::cin >> maxDeep;

checkBalanceBrackets(text, maxDeep) ? std::cout << "Результат: Соответствует\n" : std::cout << "Результат: Не соотвествует\n";

}

catch (const std::exception& ex)

{

std::cerr << ex.what() << std::endl;

return EXIT\_FAILURE;

}

return EXIT\_SUCCESS;

}

## *Stack.h*

#ifndef STACK\_H

#define STACK\_H

#include<iostream>

#include <exception>

#include<string>

#include<regex>

#include"Functions.h"

#define SIZE\_MAX 100

bool checkInput(const std::string& value);

class StackOverflow : public std::exception

{

const std::string reason;

public:

StackOverflow() : reason("Стек переполнен\n") {};

const char\* what() const { return reason.c\_str(); }

};

class StackUnderflow : public std::exception

{

const std::string reason;

public:

StackUnderflow() : reason("Стек пуст\n") {};

const char\* what() const { return reason.c\_str(); }

};

class WrongStackSize : public std::exception

{

const std::string reason;

public:

WrongStackSize() : reason("Неверный размер стека\n") {};

const char\* what() const { return reason.c\_str(); }

};

template <typename T>

class Stack

{

public:

virtual ~Stack() {};

virtual void push(const T& value) = 0;

virtual T pop() = 0;

virtual bool isEmpty() = 0;

};

template <typename T>

class StackArray : protected Stack<T>

{

private:

size\_t size;

int top;

T\* stack;

public:

StackArray(const std::string& length)

{

if (checkInput(length) && std::stoi(length) > 0 && std::stoi(length) <= SIZE\_MAX)

{

size = std::stoi(length);

top = 0;

try

{

stack = new T[size + 1];

}

catch(...)

{

throw WrongStackSize();

}

}

else throw WrongStackSize();

}

~StackArray() { delete[] stack; };

void push(const T& value) override

{

if (top == size) throw StackOverflow();

else stack[++top] = value;

}

T pop() override

{

if (isEmpty()) throw StackUnderflow();

else return stack[top--];

}

bool isEmpty() { return top == 0 ? true : false; }

void print()

{

if (!isEmpty())

{

std::cout << "Stack: ";

for (int i = top; i > 0; --i)

std::cout << stack[i] << " ";

std::cout << "\n";

}

else std::cout << "Стек пуст\n";

}

};

#endif

## *Queue.h*

#ifndef QUEUE\_H

#define QUEUE\_H

#include<iostream>

#include <exception>

#include<string>

#include<regex>

#define SIZE\_MAX 100

bool checkInput(const std::string& value);

class QueuekOverflow : public std::exception

{

const std::string reason;

public:

QueuekOverflow() : reason("Очередь переполнена\n") {};

const char\* what() const { return reason.c\_str(); }

};

class QueueUnderflow : public std::exception

{

const std::string reason;

public:

QueueUnderflow() : reason("Очередь пуста\n") {};

const char\* what() const { return reason.c\_str(); }

};

class WrongQueueSize : public std::exception

{

const std::string reason;

public:

WrongQueueSize() : reason("Неверный размер очереди\n") {};

const char\* what() const { return reason.c\_str(); }

};

template <typename T>

class Queue

{

public:

virtual ~Queue() {};

virtual void enQueue(const T& value) = 0;

virtual T deQueue() = 0;

virtual bool isEmpty() = 0;

};

template <typename T>

class CircularQueue :protected Queue<T>

{

private:

int head;

int tail;

size\_t size;

T\* queue;

public:

CircularQueue(const std::string& length)

{

if (checkInput(length) && std::stoi(length) > 0 && std::stoi(length) <= SIZE\_MAX)

{

size = std::stoi(length);

head = 0;

tail = head;

try

{

queue = new T[size + 1];

}

catch (...)

{

throw WrongQueueSize();

}

}

else throw WrongQueueSize();

}

~CircularQueue() override { delete[] queue; }

void enQueue(const T& value) override

{

if (tail + 1 <= size)

queue[tail++] = value;

else throw QueuekOverflow();

};

T deQueue() override

{

if (!isEmpty())

{

T result = queue[head];

for (int i = 0; i < tail; ++i)

std::swap(queue[i], queue[i + 1]);

--tail;

return queue[result];

}

else throw QueueUnderflow();

};

bool isEmpty() override { return head == tail; };

void print()

{

if (!isEmpty())

{

std::cout << "Queue: ";

for (int i = head; i < tail; ++i)

std::cout << queue[i] << " ";

std::cout << "\n";

}

else std::cout << "Очередь пуста\n";

}

};

#endif

## *Functions.h*

#ifndef FUNCTIONS\_H

#define FUNCTIONS\_H

#include<string>

#include "Stack.h"

bool checkBalanceBrackets(const std::string& text, const int maxDeep);

bool checkInput(const std::string& value);

#endif

## *Functions.cpp*

#include "Functions.h"

bool checkBalanceBrackets(const std::string& text, const int maxDeep)

{

StackArray <char> stack(std::to\_string(2\*maxDeep));

for (char symbol : text)

stack.push(symbol);

int round\_brackets = 0;

int square\_brackets = 0;

int figure\_brackets = 0;

char symbol = ' ';

while (!stack.isEmpty())

{

symbol = stack.pop();

switch (symbol)

{

case')':

if (square\_brackets < 0 || figure\_brackets < 0) return false;

else { round\_brackets++; break; }

case'(':

if (round\_brackets == 0) return false;

else

{

round\_brackets--;

if (round\_brackets == 0 && (square\_brackets != 0 || figure\_brackets != 0)) return false;

break;

}

case ']':

if (round\_brackets < 0 || figure\_brackets < 0) return false;

else { square\_brackets++; break; }

case'[':

if (square\_brackets == 0) return false;

else

{

square\_brackets--;

if (square\_brackets == 0 && (round\_brackets != 0 || figure\_brackets != 0)) return false;

break;

}

case'}':

if (square\_brackets < 0 || round\_brackets < 0) return false;

else { figure\_brackets++; break; }

case'{':

if (figure\_brackets == 0) return false;

else

{

figure\_brackets--;

if (figure\_brackets == 0 && (round\_brackets != 0 || square\_brackets != 0)) return false;

break;

}

default: break;

}

}

return round\_brackets ==0 && square\_brackets==0 && figure\_brackets ==0;

}

bool checkInput(const std::string& value)

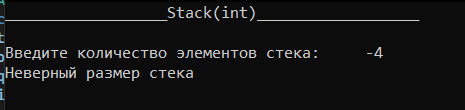
{

std::regex regular("-?\\d+");

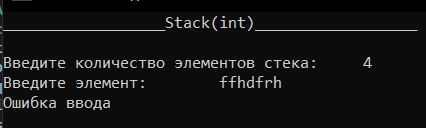
return std::regex\_match(value, regular) ? true : false;

}

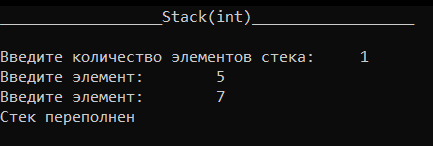
# Приложение A



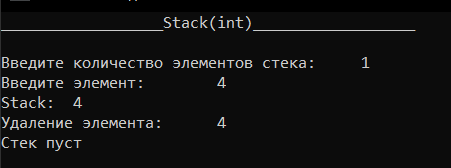
*Неверный размер структуры*



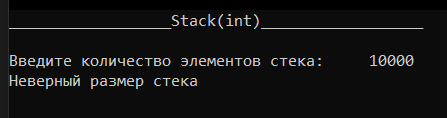
*Неверный тип элемента структуры*



*Переполнение структуры*

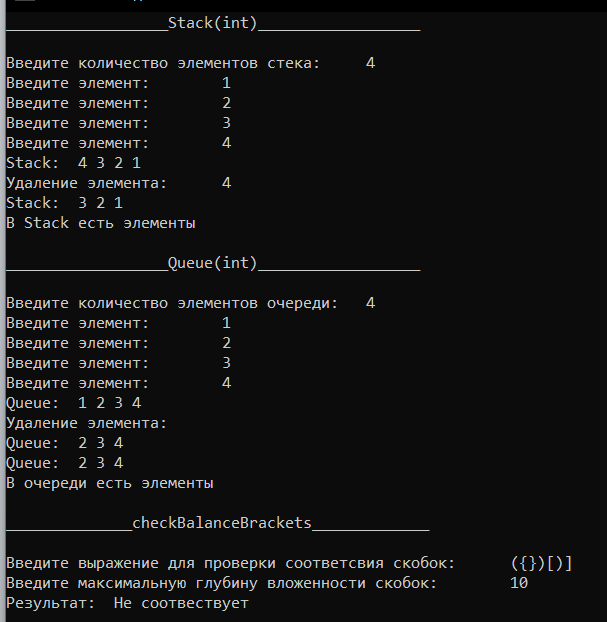


*Попытка удаления элемента из пустой структуры*



*Размер стека(очереди) превышает максимально возможный*

# Приложение B



*Корректная работа программы*

# Вывод

В ходе работы над программой были изучены:

1. Структуры данных стек и очередь, написаны шаблоны данных классов, реализованы структуры данных ограниченный стек ( на основе статического массива) и кольцевая очередь (на основе статического массива)
2. Написаны классы StackOverflow, StakUnderflow, WrongStackSize, QueueOverflow, QueueUnderflow, QueueStackSize для обработки ошибок, которые могут возникнуть при работе с данными структурами.
3. Написана функция checkBalanceBrackets на основе ограниченного стека, проверяющая соответствие скобок в заданном выражении.