Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе №19**

Дисциплина: «Основы теории алгоритмов и структуры данных»

Тема: “ Основные алгоритмы работы со стеками”

Выполнил:

Студент группы ИВТ-20-2Б Галинов О.Ю.

Проверила:

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2021

**Постановка задачи**

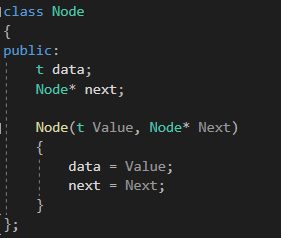
Написать собственный класс, отражающий работу стека.

**Анализ задачи**

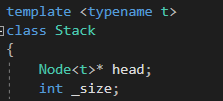
**1.** Для решения задачи необходимо:

**1.1.** Организовать шаблонный класс Node с полями data типа t и указателем next типа \*Node.

А также конструктор данного класса.

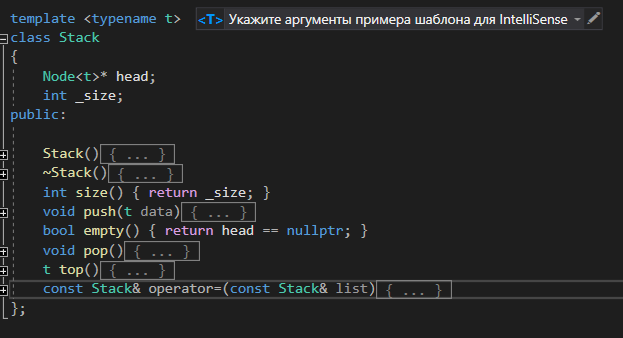


**1.2.** Организовать перегруженный класс Stack с полями \_size типа int, и указателем head типа Node<T>

.

**2.** В ходе работы были использованы следующие типы данных:

**2.1.** Перегруженный класс Stack с полями \_size типа int, и указателем head типа Node<T>в заголовочном файле Stack.h.



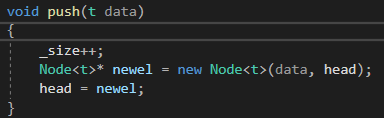
**2.2.** Элементы класса Stack типа int.

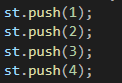




**3.** Для решения задачи данные были представлены в следующем виде:

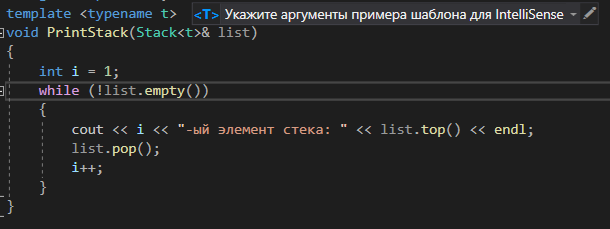
**3.1.** Стек заполняется через метод класса, в который передается значение типа t.





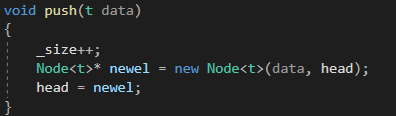
**4.** Для операций ввода и вывода использовались следующие операторы и функции:

**4.1.** Для вывода стека используется функция void PrintStack (Stack<t>& list), в которую передается сам стек типа t.

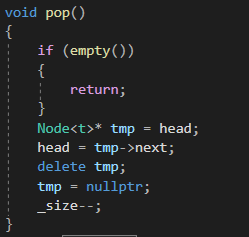


**5.** Поставленные задачи будут решены следующими действиями:

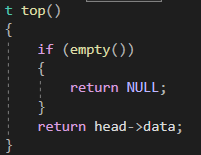
**5.1.** В перегруженном методе push() в качестве параметра передаётся параметр, который необходим добавить в стек. Создаётся указатель newel, далее вызывается конструктор класса node, в который передается элемент, добавляющийся в стек и указатель на вершину стека, в конце вершине присваивается значение того элемента, который был добавлен.



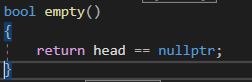
**5.2.** В перегруженном методе pop() создаётся указатель tmp, указывающий на вершину стека. А вершине присваивается tmp, который указывается на след элемент. Далее указатель tmp удаляется, а его значению присваивается nullptr, size уменьшается на 1.



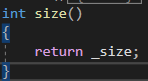
**5.3.** В перегруженном методе top() возвращается элемент, который находится в head->data.



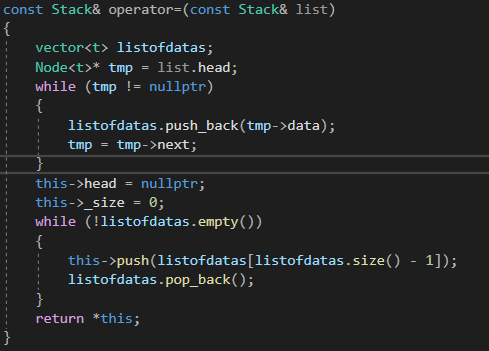
**5.4.** В перегруженном методе empty() возвращается true/false в зависимости от того пуст ли стек: если первый элемент стека – head, указывает на nullptr, то стек пуст, иначе заполнен.



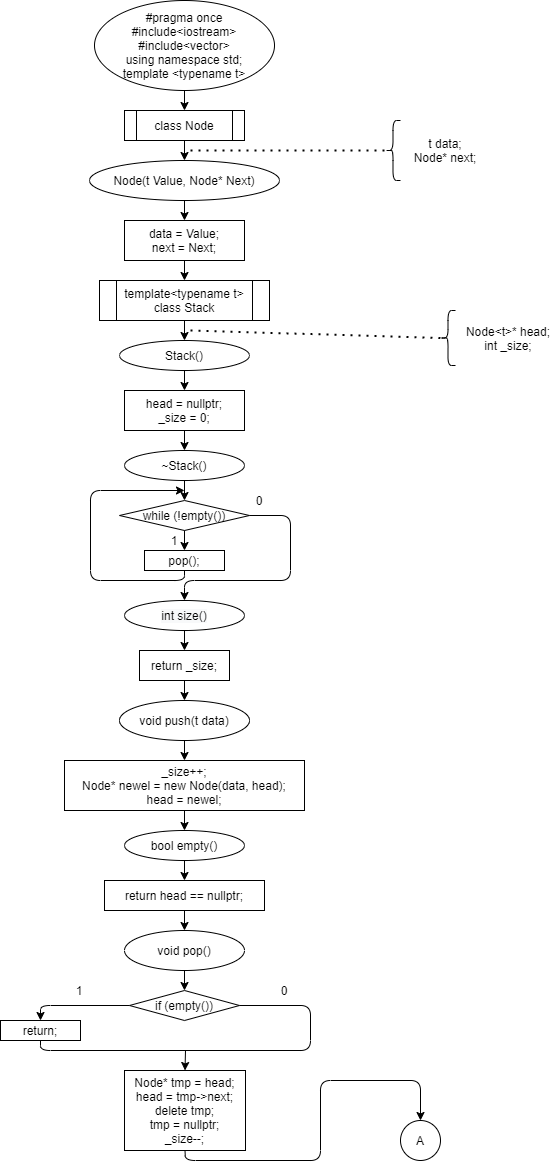
**5.5.** В перегруженном методе size() возвращается значение поля \_size.

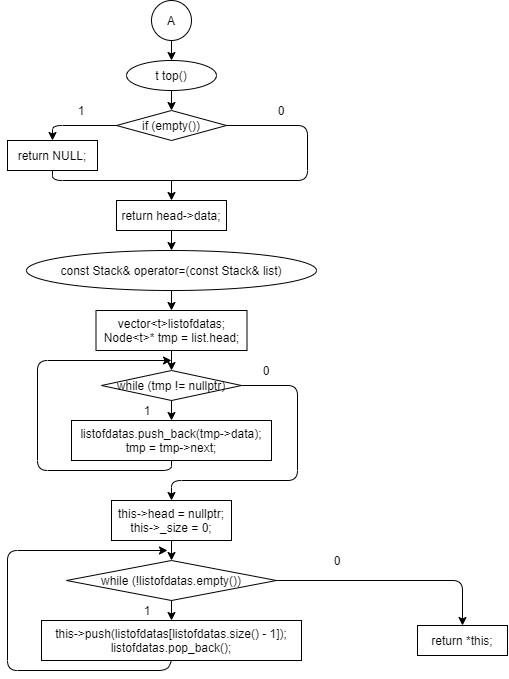


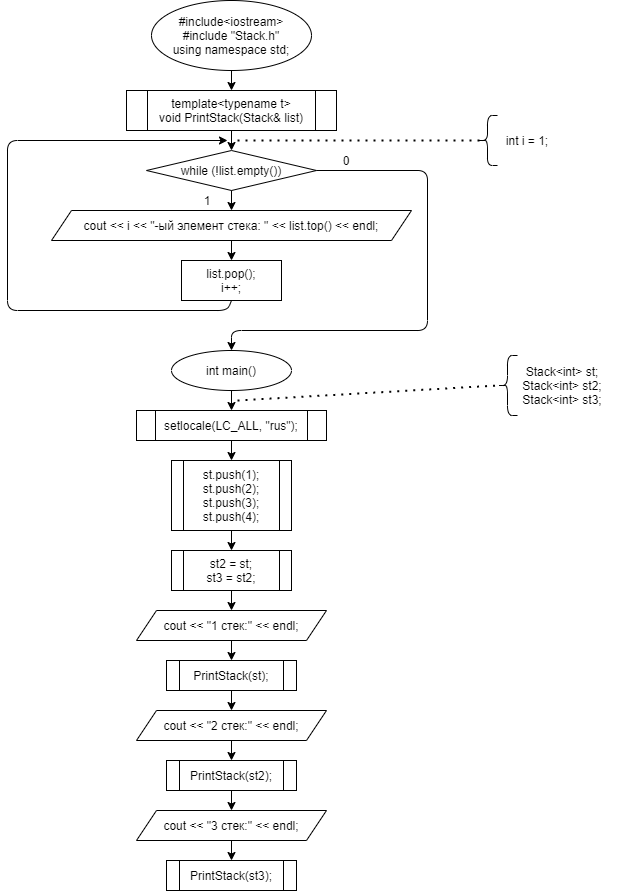
**5.6.** В перегруженном операторе = описывается метод приравнивания одного стека к другому.



**Блок-схема**

****

****

****

**Код**

#include <iostream>

#include "Stack.h"

using namespace std;

template <typename t>

void PrintStack(Stack<t>& list)

{

int i = 1;

while (!list.empty())

{

cout << i << "-ый элемент стека: " << list.top() << endl;

list.pop();

i++;

}

}

int main()

{

setlocale(LC\_ALL, "rus");

Stack<int> st;

st.push(1);

st.push(2);

st.push(3);

st.push(4);

Stack<int> st2;

st2 = st;

Stack<int> st3;

st3 = st2;

cout << "1 стек:" << endl;

PrintStack(st);

cout << endl << "2 стек:" << endl;

PrintStack(st2);

cout << endl << "3 стек:" << endl;

PrintStack(st3);

}

#pragma once

#include <iostream>

#include <vector>

using namespace std;

template <typename t>

class Node

{

public:

t data;

Node\* next;

Node(t Value, Node\* Next)

{

data = Value;

next = Next;

}

};

template <typename t>

class Stack

{

Node<t>\* head;

int \_size;

public:

Stack()

{

head = nullptr;

\_size = 0;

}

~Stack()

{

while (!empty())

pop();

}

int size()

{

return \_size;

}

void push(t data)

{

\_size++;

Node<t>\* newel = new Node<t>(data, head);

head = newel;

}

bool empty()

{

return head == nullptr;

}

void pop()

{

if (empty())

{

return;

}

Node<t>\* tmp = head;

head = tmp->next;

delete tmp;

tmp = nullptr;

\_size--;

}

t top()

{

if (empty())

{

return NULL;

}

return head->data;

}

const Stack& operator=(const Stack& list)

{

vector<t> listofdatas;

Node<t>\* tmp = list.head;

while (tmp != nullptr)

{

listofdatas.push\_back(tmp->data);

tmp = tmp->next;

}

this->head = nullptr;

this->\_size = 0;

while (!listofdatas.empty())

{

this->push(listofdatas[listofdatas.size() - 1]);

listofdatas.pop\_back();

}

return \*this;

}

};

**Скриншоты**

