Московский государственный технический

университет им. Н.Э. Баумана

Проверил:

Преподаватель каф. ИУ5

Подпись и дата:

Факультет «Информатика и система управления»

Кафедра ИУ5 «Система обработки информации и управления»

Выполнил:

Студент группы ИУ5-22Б

Яншин Т. И.

Подпись и дата:

Курс «Программирование на основе классов и шаблонов»

Отчёт по лабораторной работе №4

Шаблоны классов

## **Задание**

## *Дано описание класса MyStack (Приложение 1, файл MyStack.h), который реализует на основе односвязного списка динамическую структуру данных типа стек.*

## *Разработайте реализацию класса MyStack.*

## *Разработайте функцию (глобальную), которая выполняет разложение на простые множители целого числа N. Для хранения множителей функция должна использовать класс MyStack. Прототип функции: void Multipliers(int n, MyStack<DATA> &stack).*

## *В функции main() распечатайте множители, которые функция Multipliers() записывает в стек, сначала по убыванию, а потом по возрастанию. Например, для N=3960 программа должна вывести:*

## *3960=11 \* 5 \* 3 \* 3 \* 2 \* 2 \* 2*

## *3960=2 \* 2 \* 2 \* 3 \* 3 \* 5 \* 11*

## *Классы ListNode и MyStack необходимо упаковать в самостоятельную статическую библиотеку.*

Стек представляет собой динамическую структуру (то-есть структуру, размер которой может изменяться в процессе выполнения программы), предназначенную для временного хранения данных некоторого типа INF, который может быть как базовым, так и определяемым пользователем. Запись данных в стек и выборка данных из стека производятся путем обращения к его вершине.

В данной работе стек реализуется на базе односвязного списка. При записи в стек (операция PUSH) в начало списка добавляется узел, а при выборке из стека (операция POP) узел удаляется из начала списка. Чтобы получить доступ к следующей ячейке стека нужно удалить предыдущую. При выполнении операции POP данные из стека не считываются. Данные могут считываться только из вершины стека. Для чтения данных используется специальная функция, которая читает данные без удаления узла из вершины стека.

Для обеспечения доступа к данным, хранящимся в узлах типа ListNode, класс MyStack сделан дружественным по отношению к классу ListNode. В этом случае все методы класса MyStack получают доступ к скрытым данным класса ListNode.

Чтобы в узлах можно было бы хранить данные различных типов и чтобы узлы класса ListNode можно было бы использовать в различных структурах (например, для реализации списка или очереди), класс ListNode реализован в виде шаблона семейства классов с двумя формальными параметрами: типом хранимых данных (class INF) и дружественным классом, реализующим некоторую структуру данных, например стек (class FRIEND).

Методы шаблонного класса не должны зависеть от значений формальных параметров и должны быть одинаковыми для всех типов хранимых данных и дружественных классов. Для реализации этого требования в лабораторной работе класс ListNode сделан закрытым (то-есть в нем нет методов в разделе public:, хотя могли бы и быть), а доступ к его элементам осуществляется через интерфейс дружественного класса FRIEND, являющегося одним из двух формальных параметров класса ListNode (в данной работе это MyStack<INF>).

## **Текст программы**

Внешний CMakeLists.txt

cmake\_minimum\_required (VERSION 3.8)

set(project "lab4")

project(**${project}**)

set(CMAKE\_CXX\_STANDARD 17)

set(**${project}**\_SOURCES

main.cpp)

add\_subdirectory(MyStack)

set(**${project}**\_SOURCE\_LIST

${**${project}**\_SOURCES})

add\_executable(**${project}**

${**${project}**\_SOURCE\_LIST} )

target\_link\_libraries(**${project}** MyStack)

Внутренний MyStack /CMakeLists.txt

cmake\_minimum\_required(VERSION 3.8)

set(project "MyStack")

project(**${project}**)

set(CMAKE\_CXX\_STANDARD 17)

set(**${project}**\_SOURCES

MyStack.cpp)

set(**${project}**\_HEADERS

MyStack.h)

set(**${project}**\_SOURCE\_LIST

${**${project}**\_SOURCES}

${**${project}**\_HEADERS})

add\_library(**${project}**

STATIC

${**${project}**\_SOURCE\_LIST})

Main.cpp

#include "MyStack/MyStack.h"

using namespace std;

static void Multipliers(int n, MyStack<int>& stack)

{

int temp = n;

int i = 2;

while (i \* i <= n)

{

if (n % i == 0)

{

stack.push(i);

n /= i;

}

else

{

i++;

}

}

if (n >= 1)

{

stack.push(n);

}

}

int main()

{

setlocale(LC\_ALL, "Rus");

MyStack <char> stack;

MyStack <char> stack2;

MyStack <int> stack4;

stack.push('a');

stack.push('b');

stack.push('c');

stack2 = stack;

MyStack stack3(stack);

int n = 0;

while (!(std::cin >> n) or (n <= 0))

{

std::cout << "Ошибка ввода: \n";

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

}

Multipliers(n, stack4)

stack.PrintStack();

stack2.PrintStack();

stack3.PrintStack();

stack4.PrintStackM(n);

stack3.PrintStackReverse();

cout << endl;

stack4.PrintStackReverseM(n);

return 0;

}

MyStack.h

#include <iostream>

using namespace std;

#ifndef MyStack\_h // защита от повторной компиляции

#define MyStack\_h // модуль подключен

// Шаблонный класс ListNode (узел односвязного списка)

template<class INF, class FRIEND>

class ListNode // узел списка

{

private:

INF d; // информационная часть узла

ListNode\* next; // указатель на следующий узел списка

public:

ListNode() // конструктор

{

d = INF();

next = nullptr;

}

friend FRIEND;

};

// Шаблонный класс MyStack на основе односвязного списка.

template<class INF>

class MyStack

{

typedef class ListNode<INF, MyStack<INF>> Node;

Node\* top;

public:

MyStack() // конструктор

{

this->top = nullptr;

}

MyStack(MyStack<INF>& currStack)

{

this->top = new Node;

Node\* head = this->top;

Node\* head2 = currStack.top;

while (head2 != nullptr)

{

if (head2->next != nullptr)

{

head->next = new Node;

}

head->d = head2->d;

head = head->next;

head2 = head2->next;

}

}

~MyStack() // освободить динамическую память

{

while (!empty())

{

this->pop();

}

~~delete this->top;~~

}

bool empty() // стек пустой?

{

if (this->top == nullptr)

{

return true;

}

return false;

}

void push(INF n) // добавить узел в вершину стека

{

Node\* newNode = new Node;

newNode->d = n;

newNode->next = this->top;

this->top = newNode;

}

void pop() // удалить узел из вершины стека

{

Node\* tempNode = this->top->next;

delete this->top;

this->top = tempNode;

}

INF top\_inf() // считать информацию из вершины стека

{

return this->top->d;

}

void PrintStack()

{

Node\* tempNode = this->top;

while (tempNode != nullptr)

{

//tempNode = this->top;

cout << tempNode->d;

if (tempNode->next != nullptr)

{

cout << " ";

}

tempNode = tempNode->next;

}

cout << endl;

}

void PrintStackM(int n)

{

Node\* tempNode = this->top;

std::cout << n << " = ";

while (tempNode != nullptr)

{

//tempNode = this->top;

cout << tempNode->d;

if (tempNode->next != nullptr)

{

cout << " \* ";

}

tempNode = tempNode->next;

}

cout << endl;

}

void PrintStackReverse()

{

if (this->empty())

{

return; //если стек пуст

}

auto bottomElement = this->top->d; // Сохраняем верхний элемент стека

this->pop(); // Удаляем верхний элемент стека

PrintStackReverse(); // Рекурсивно обрабатываем оставшийся стек

cout << bottomElement << " "; // Выводим сохраненный элемент

this->push(bottomElement); // Возвращаем удаленный элемент обратно в стек

}

void PrintStackReverseM(int n)

{

static bool flag = 1;

if (flag)

{

cout << n << " = ";

flag = 0;

}

if (this->empty())

{

return; // если стек пуст

}

int bottomElement = this->top->d; // Сохраняем верхний элемент стека

this->pop(); // Удаляем верхний элемент стека

if (!this->empty())

{

PrintStackReverseM(n); // Рекурсивно обрабатываем оставшийся стек

cout << " \* ";

}

cout << bottomElement; // Выводим сохраненный элемент

this->push(bottomElement); // Возвращаем удаленный элемент обратно в стек

}

MyStack& operator=(MyStack& currStack)

{

if (this == &currStack)

{

return \*this;

}

// Освобождаем память, занимаемую текущим стеком

while (!empty())

{

this->pop();

}

Node\* head = this->top;

Node\* head2 = currStack.top;

while (head2 != nullptr)

{

Node\* newNode = new Node;

newNode->d = head2->d;

if (head == nullptr)

{

this->top = newNode;

}

else

{

head->next = newNode;

}

head = newNode;

head2 = head2->next;

}

return \*this;

}

#endif

Fraction.cpp

#include "MyStack.h"

**Тестирование**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |