**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра ИУ5. Курс «Программирование на основе классов и шаблонов»

Отчет по лабораторной работе №4

«Шаблоны классов.

Использование шаблонного класса MyStack для хранения простых множителей

целых чисел»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-21 |  | преподаватель каф. ИУ5 |
| Агличеев Михаил |  | Селиверстова А.В. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2020 г.

1. Постановка задачи.

Стек представляет собой динамическую структуру (то-есть структуру, размер которой можети зменяться 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 изначала 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 методовв 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&lt; INF&gt;).

Указания:

1. Пример внешнего определения методов шаблонного класса:

template &lt;class INF&gt;

bool MyStack&lt; INF&gt;::empty(void)

{

if(top==NULL)

return true;

else

return false;

}

2. Определение (переименование) типа узла ListNode, хранящего данные класса INF и

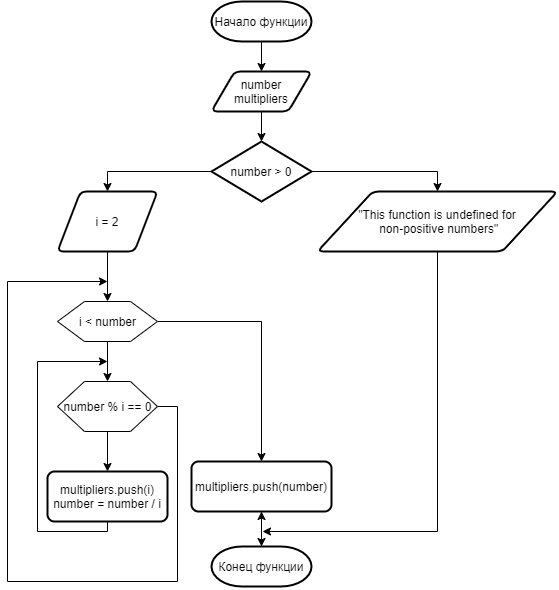
использующего в качестве дружественного класса FRIEND шаблонный класс MyStack&lt;INF&gt; :

typedef class ListNode &lt; INF, MyStack &lt;INF&gt; &gt; Node;

Теперь вместо class ListNode &lt; INF, MyStack &lt;INF&gt; &gt; он будет называться просто Node.

1. Разработка алгоритма

Блок-схема алгоритма нахождения множителей числа:



1. Текст программы

Файл my\_stack.h:

#ifndef MY\_STACK\_H

#define MY\_STACK\_H

#include "pch.h"

#include "framework.h"

//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

ListNode(void) { next = NULL; } //конструктор

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(void); // конструктор

~MyStack(void); // освободить динамическую память

bool empty(void); // стек пустой?

bool push(INF n); // добавить узел в вершину стека

bool pop(void); // удалить узел из вершины стека

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

};

template <class INF>

MyStack<INF>::MyStack(void) {

top = nullptr;

}

template <class INF>

MyStack<INF>::~MyStack(void) {

while (top != nullptr) {

Node\* temp = top;

top = top->next;

delete temp;

}

}

template <class INF>

bool MyStack<INF>::empty(void) {

return (top == nullptr);

}

template <class INF>

bool MyStack<INF>::push(INF data) {

Node\* node = new Node();

node->next = top;

node->d = data;

top = node;

return true;

}

template <class INF>

bool MyStack<INF>::pop(void) {

if (!empty()) {

top = top->next;

return true;

}

return false;

}

template <class INF>

INF MyStack<INF>::top\_inf(void) {

return top->d;

}

#endif // MY\_STACK\_H

Файл multipliers.h:

#ifndef MULTIPLIERS\_H

#define MULTIPLIERS\_H

#include "..\Stack\my\_stack.h"

int main();

void get\_multipliers(int, MyStack<int>&);

#endif // MULTIPLIERS\_H

Файл multipliers.cpp:

#include <iostream>

#include "multipliers.h"

using namespace std;

int main() {

int number;

cout << "Please input a number to get it's multipliers: ";

cin >> number;

cout << endl;

if (number <= 0) {

cout << "Multipliers of non-positive numbers are undefined!" << endl;

system("pause");

return 0;

}

MyStack<int> multipliers;

MyStack<int> other;

get\_multipliers(number, multipliers);

cout << number << "=";

while (!multipliers.empty()) {

other.push(multipliers.top\_inf());

cout << multipliers.top\_inf() << "\*";

multipliers.pop();

}

cout << '\b' << '\0' << '\b' << endl;

cout << number << "=";

while (!other.empty()) {

cout << other.top\_inf() << "\*";

other.pop();

}

cout << '\b' << '\0' << '\b' << endl;

system("pause");

return 1;

}

void get\_multipliers(int number, MyStack<int>& multipliers) {

if (number <= 0) {

cout << "This function is undefined for numbers less that or equal to 0!" << endl << endl;

return;

}

for (int i = 2; i < number; i++) {

while (number % i == 0) {

multipliers.push(i);

number /= i;

}

}

multipliers.push(number);

}

5. Анализ результатов

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

Как видно, представленный алгоритм успешно справляется с поиском множителей, причём работает даже с простыми числами и в предельном случае, когда вводимое число – единица.