**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Отчет по лабораторной работе №4

«Шаблоны классов. Использование шаблонного класса MyStack для хранения простых множителей целых чисел»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Выполнил:** | |  | **Принял:** | |
| ФИО: | \_Цыпышев Т. А.\_\_\_\_\_ |  | ФИО: | \_Правдина А. Д.\_\_\_\_\_ |
| Группа: | \_ИУ5-21Б\_\_\_\_\_\_\_\_\_\_\_ |  | Должность: | \_Преподаватель\_\_\_\_\_ |
| Дата: | \_07.05.2023\_\_\_\_\_\_\_\_\_ |  | Дата: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Подпись: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | Подпись: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Москва, 2023 г.

**Постановка задачи**

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

1. Разработайте реализацию интерфейса класса в виде файла MyStack.cpp.

2. 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).

3. 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

**Текст программы**

**main.cpp**

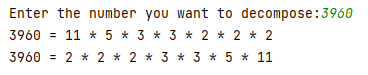
*/// Copyright 2022 ttsypyshev <ttsypyshev01@gmail.com>*#include "..\include\MyStack.h"  
  
#include "iostream"  
  
template<class INF>  
MyStack<INF>::MyStack() {  
 top = nullptr;  
}  
  
template<class INF>  
MyStack<INF>::MyStack(const MyStack<INF> &other) {  
 top = nullptr;  
 Node \*current = other.top;  
 MyStack<INF> tmp;  
 while (current != nullptr) {  
 tmp.push(current->d);  
 current = current->next;  
 }  
 Node \*tmp\_node = tmp.top;  
 while (tmp\_node != nullptr) {  
 push(tmp\_node->d);  
 tmp\_node = tmp\_node->next;  
 }  
}  
  
template<class INF>  
MyStack<INF> &MyStack<INF>::operator=(const MyStack<INF> &other) {  
 if (this != &other) {  
 while (!empty()) {  
 pop();  
 }  
 Node \*current = other.top;  
 MyStack<INF> tmp;  
 while (current != nullptr) {  
 tmp.push(current->d);  
 current = current->next;  
 }  
 Node \*tmp\_node = tmp.top;  
 while (tmp\_node != nullptr) {  
 push(tmp\_node->d);  
 tmp\_node = tmp\_node->next;  
 }  
 }  
 return \*this;  
}  
  
template<class INF>  
MyStack<INF>::~MyStack() {  
 while (!empty()) pop();  
}  
  
template<class INF>  
bool MyStack<INF>::empty() {  
 return top == nullptr;  
}  
  
template<class INF>  
bool MyStack<INF>::push(INF value) {  
 Node \*newNode = new Node;  
 newNode->d = value;  
 newNode->next = top;  
 top = newNode;  
 return true;  
}  
  
template<class INF>  
bool MyStack<INF>::pop() {  
 if (empty()) {  
 std::cout << "Stack is empty!" << std::endl;  
 return false;  
 }  
 Node \*oldNode = top;  
 top = top->next;  
 delete oldNode;  
}  
  
template<class INF>  
INF \*MyStack<INF>::top\_inf() {  
 if (empty()) {  
 std::cout << "Stack is empty!" << std::endl;  
 return nullptr;  
 }  
 return &top->d;  
}  
  
void Multipliers(int n, MyStack<int> &stack) {  
 int i = 2;  
 while (n > 1) {  
 if (n % i == 0) {  
 stack.push(i);  
 n /= i;  
 } else {  
 i++;  
 }  
 }  
}  
  
int main() {  
 int n = 3960;  
*// int n;  
// std::cout << "Enter the number you want to decompose:";  
// std::cin >> n;* MyStack<int> stack1, stack2;  
 Multipliers(n, stack1);  
  
 if (stack1.empty()) {  
 std::cout << "This number cannot be decomposed into prime numbers!" << std::endl;  
 return 1;  
 }  
  
 std::cout << n << " = ";  
 while (!stack1.empty()) {  
 std::cout << \*stack1.top\_inf();  
 stack2.push(\*stack1.top\_inf());  
 stack1.pop();  
 if (!stack1.empty()) {  
 std::cout << " \* ";  
 }  
 }  
 std::cout << std::endl;  
  
 std::cout << n << " = ";  
 while (!stack2.empty()) {  
 std::cout << \*stack2.top\_inf();  
 stack2.pop();  
 if (!stack2.empty()) {  
 std::cout << " \* ";  
 }  
 }  
 std::cout << std::endl;  
  
 return 0;  
}

**MyStack.h**

*/// Copyright 2022 ttsypyshev <ttsypyshev01@gmail.com>  
  
///Шаблонный класс MyStack на основе односвязного списка.*#ifndef **MyStack\_h** *// защита от повторной компиляции*#define **MyStack\_h** *// модуль подключен  
//#include "Data.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 = nullptr; } *//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(const MyStack<INF> &other); *// конструктор копирования* ~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* MyStack<INF> &operator=(const MyStack<INF> &other); *// оператор присваивания*};  
  
#endif

**Тестирование функций**

****

****

****