**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Программирование на основе классов и шаблонов»

Отчет по лабораторной работе №2

«Реализация класса стек»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-22 |  | преподаватель каф. ИУ5 |
| Барышников Михаил |  | Козлов А.Д. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2020 г.

**Задача**

1. Реализовать класс стек – абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO (last in, first out – последним пришел, первым вышел)
2. Разработать функцию, которая выполняет разложение на простые множители целого числа N. Для хранения множителей функция должна использовать класс стек.
3. В функции main( ) распечатайте множители, которые функция 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

1. Для переворота последовательности создать метод reverse();

**Текст программы**

* class\_stack.h

#pragma once

#include <iostream>

class Stack

{

struct element

{

int simple\_num;

element\* previos\_elem;

};

element\* top;

public:

Stack();

Stack(Stack&);

~Stack();

void push(int);

int pop();

bool empty();

void reverse();

Stack operator=(int);

friend std::ostream &operator<<(std::ostream&, Stack&);

friend std::istream &operator>>(std::istream&, Stack&);

};

* class\_stack.cpp

#include "class\_stack.h"

Stack Stack::operator=(int b)

{

if (b > 1)

{

register int i = 2;

for (; b > 1;)

if (b % i)

++i;

else if (b /= i)

push(i);

}

else

{

push(b);

}

return \*this;

}

Stack::Stack() : top(nullptr) {}

Stack::Stack(Stack &b)

{

Stack n;

element \*last = b.top;

top = nullptr;

while (last)

{

n.push(last->simple\_num);

last = last->previos\_elem;

}

while (!n.empty())

{

push(n.pop());

}

}

Stack::~Stack()

{

while (top)

{

element\* del = top;

top = top->previos\_elem;

delete del;

}

}

void Stack::push(int b)

{

element \*next = new element;

next->simple\_num = b;

next->previos\_elem = top;

top = next;

}

int Stack::pop()

{

if (empty())

return -1;

int buf = top->simple\_num;

element\* del = top;

top = top->previos\_elem;

delete del;

return buf;

}

bool Stack::empty()

{

return top ? false : true;

}

void Stack::reverse()

{

Stack \*b = new Stack;

while (!empty())

{

b->push(pop());

}

\*this = \*b;

}

std::ostream &operator<<(std::ostream& stream, Stack& b)

{

while (b.top)

stream << b.pop() << ' ';

return stream;

}

std::istream & operator>>(std::istream &stream, Stack &b)

{

int i;

stream >> i;

b = i;

return stream;

}

* main.cpp

#include "class\_stack.h"

int main()

{

Stack a;

int num;

while (true)

{

std::cout << "Enter number: ";

std::cin >> num;

a = num;

std::cout << "Simple factors:";

std::cout << "\n" << a << "\n";

a = num;

a.reverse();

std::cout << "Reverse simple factors:";

std::cout << "\n" << a << "\n";

}

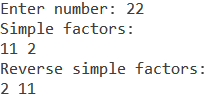
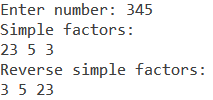
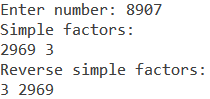
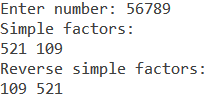
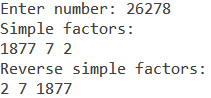
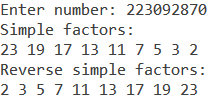
system("pause");

return 0;

}

**Анализ результатов**

Для всех приведенных примеров программа верно разложила числа на простые множители:

1. 
2. 
3. 
4. 
5. 
6. 

В классе реализованы следующие методы:

* push(); - поместить элемент в стек
* pop(); - удалить последний элемент из стека
* empty(); - стек пустой?
* reverse(); - развернуть стек
* operator=(); - присвоить элементу класса стек значение
* operator<<(); - оператор вывода для элемента класса стек
* operator>>(); - ввод в стек значения из потока

**Вывод**

В результате написания программы был получен опыт работы с классами (ООП). Был реализован класс стек. Стек представляет собой динамическую структуру, предназначенную для временного хранения данных некоторого типа, 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. Еще были переопределены операторы вывода << и ввода >>, для этого создана friend – функция класса.