**Лабораторная работа №4 Черкасов Александр Андреевич 1 вариант А-08-19**

**Содержание**

[**Задание (4.1)** 1](#_Toc53335864)

[**Задание (4.2)** 1](#_Toc53335865)

[**Задание (4.3)** 1](#_Toc53335866)

[**1. Постановка задачи** 1](#_Toc53335867)

[**2. Разработка программы** 2](#_Toc53335868)

[2.1 Разработка функций и методов классов 2](#_Toc53335869)

[2.2 Разработка интерфейса 4](#_Toc53335870)

[**3. Реализация и тестирование программы** 4](#_Toc53335871)

[3.1 Описание разработанной программы 4](#_Toc53335872)

[3.2 Тестирование программы 5](#_Toc53335873)

[**Вывод** 8](#_Toc53335874)

[**Приложение. Код программы** 8](#_Toc53335875)

# **Задание (4.1)**

Создать класс целых чисел. Определить оператор ++, как функцию-член и -- как дружественную функцию

# **Задание (4.2)**

Создать объект очередь с перегруженными +, +=, добавление элемента в очередь и сложение очередей, -- для извлечения из очереди, - для вычитания очередей.

# **Задание (4.3)**

Создать класс вектор, содержащий ссылку на long, размерность вектора и переменную ошибки. Класс имеет конструкторы по умолчанию, конструктор с одним и двумя параметрами, конструктор копирования и деструктор. Определить оператор +, - , \* , - как дружественные функции, =, +=, -=, \*=,[] - как функции-члены. Определить операторы =, +, - , \*, +=, -=, \*= с числом типа long, операторы ++ и --. Определить функцию печати. Перегрузить операторы вывода и ввода в поток.

# **1. Постановка задачи**

Разработать объектно-ориентрованную программу в соответствии с заданиями 4.1, 4.2 и 4.3

**4.1**

**Функции**: Ввод и вывод данного целого числа, реализация методов по его увеличению и уменьшению через операторы ++ и – как префиксное, так и постфиксное

**Входные данные**: ничего/начальное значение

**Выходные данные**: Текущее значение

**Ограничения**: Значение – целое число, лежащее в диапазоне [-2147483648; 2147483647]

**4.2**

**Функции**: Реализация методов добавления, извлечения элементов в очереди, сложения и вычитания очередей

**Входные данные**: Объекты для очереди

**Выходные данные**: Текущее состояние очереди

**Ограничения**: Объекты очереди должны соответствовать своему типу

**4.3**

**Функции**: Реализация методов добавления и извлечения элементов в векторе, умножения элементов вектора на число, сложения и вычитания векторов, получения элемента вектора под определенным индексом

**Входные данные**: Элементы вектора по одиночке/ Размер и такое же количество элементов вектора

**Выходные данные**: Текущий вектор

**Ограничения**: Элементы вектора – целые числа типа long в диапазоне [-2147483648; 2147483647]

**Вид приложения** - консольное приложение на языке C++.

**Среда разработки** – CLion

# 

# **2. Разработка программы**

2.1 Разработка функций и методов классов

**4.1** Необходимо создать класс tInt, содержащий в себе методы увеличения и уменьшения текущего значения, а также получения значения.

**4.2** Необходимо создать класс queue, содержащий в себе методы добавления и извлечения элементов, сложения и вычитания очередей, присваивания.

**4.3** Необходимо создать класс vector, содержащий в себе методы добавления и извлечения элементов, обращения к элементу по индексу, сложения и вычитания векторов, умножения его элементов на число, присваивания, ввода и вывода. Так как имеем дело с динамическими массивами, необходимы функции переопределения выделенной памяти и копирования массивов.

Поля, методы, конструкторы приведены в таблицах 2.1, 2.2, 2.3

Табл 2.1 tInt

|  |  |
| --- | --- |
| **tInt** | |
| **Конструкторы** | **Описание** |
| tInt(); | Стандартный констуктор. Устанавливает начальное значение в 0 |
| explicit tInt(const int&); | Параметризированный констукторю Устанавливает начальное значение по входному |
| **Поля** |  |
| int\* value; | Хранимое текущее значение |
| **Деструктор** |  |
| ~tInt() | Деструктор, подчищающий выделенную память |
| **Методы** |  |
| tInt operator++(); | Префиксное увеличение числа на 1. Возвращает измененное значение |
| tInt operator++(int); | Префиксное увеличение числа на 1. Возвращает предыдущее значение |
| **Дружественные методы** |  |
| friend tInt& operator--(tInt&); | Префиксное уменьшение числа на 1. Возвращает измененное значение |
| friend tInt& operator--(tInt&,int); | Префиксное уменьшение числа на 1. Возвращает предыдущее значение |
| friend std::ostream& operator<<(std::ostream&,const tInt&); | Перегруженный оператор вывода в поток текущего значение |

Табл 2.2 queue

|  |  |
| --- | --- |
| **queue** | |
| **Конструкторы** | **Описание** |
| queue() | Конструктор по умолчанию. Инициация объекта и вектора под хранение элементов |
| **Деструктор** |  |
| ~queue() | Стандартный деструктор ( = default) |
| **Поля** |  |
| std::vector<T> array; | Вектор под хранение элементов очереди |
| **Методы** | *//Заметка: T – шаблонный тип* |
| void push(const T&); | Стандартный способ добавление элемента в очередь |
| T pop(); | Удаление первого элемента из очереди. Возвращает его |
| Табл 2.2 продолжение | |
| queue& operator+=(const T&); | Добавление элемента через += |
| queue& operator--(); | Префиксное удаление первого элемента. Возвращает измененную очередь |
| queue operator--(int); | Постфиксное удаление первого элемента. Возвращает неизмененную очередь |
| queue operator+(const queue&); | Сложение очередей |
| queue operator-(const queue&); | Вычитание очередей |
| queue operator=(const queue&); | Присваивание очередей |
| **Дружественные методы** |  |
| friend std::ostream& operator<<(std::ostream&,const queue&); | Перегруженный оператор вывода очереди в поток.  Формат вывода: [ ‘элементы через пробел’ ] |

Табл 2.3 vector

|  |  |
| --- | --- |
| **vector** | |
| **Конструкторы** | **Описание** |
| vector(); | Конструктор по умолчанию. Устанавливает текущую длину и количество элементов в 0 |
| explicit vector(const size\_t&); | Конструктор, заранее выделяющий память под количество элементов, указанное в параметре |
| vector(long\*,const size\_t&); | Конструктор, копирующий переданный массив, с заданной длиной |
| vector(const vector&); | Конструктор копирования |
| **Деструктор** |  |
| ~ vector() | Деструктор, очищающий выделенную память |
| **Поля** |  |
| long\* array; | Массив, для хранения элементов вектора |
| size\_t length; | Количество элементов, под которое выделена память |
| size\_t element\_count; | Текущее количество элементов |
| static const size\_t length\_step = 10; | «Шаг» для переопределения памяти |
| static std::invalid\_argument no\_element; | Переменная ошибки, выбрасываемой при отсутствии вычитаемого элемента в векторе |
| **Методы** |  |
| void push\_back(const long&); | Стандартный способ добавления элемента в вектор |
| long pop(); | Стандартный способ извлечения элемента из вектора |
| vector& operator=(const long&); | Оператор присваивания, создающий вектор из одного элемента |
| vector& operator+=(const long&); | Добавление элемента в вектор через += |
| vector& operator-=(const long&); | Извлечение определенного элемента из вектора через -= |
| long& operator[](const size\_t&); | Получение элемента под индексом |
| vector& operator++(); | Префиксное добавление нулевого элемента. Возвращает измененный вектор |
| vector operator++(int); | Постфиксное добавление нулевого элемента. Возвращает неизмененный вектор |
| vector& operator--(); | Префиксное извлечение элемента из вектора. Возвращает измененный вектор |
| vector operator--(int); | Постфиксное извлечение элемента из вектора. Возвращает неизмененный вектор |
| vector& operator=(const vector&); | Оператор присваивания векторов |
| void reallocate\_memory(const size\_t&); | Функция переопределения выделенной под элементы памяти |
| static void copy(const long\*,const size\_t&,long\*); | Функция копирования старого массива в новый |
| **Дружественные методы** |  |
| friend vector operator+(const vector&,const long&); | Добавление элемента в вектор через + |
| friend vector operator-(const vector&,const long&); | Извлечение определенного элемента из вектора через - |
| friend vector operator\*(const vector&,const long&); | Умножение всех элементов вектора на входное число |
| Табл 2.3 продолжение | |
| friend std::ostream& operator<<(std::ostream&,const vector&); | Вывод текущего вектора в поток.  Формат вывода: { ‘элементы через пробел’ } |
| friend std::istream& operator>>(std::istream&,vector&); | Ввод вектора через поток.  Формат: Размер, элементы |

2.2 Разработка интерфейса

Необходимо предоставить пользователю диалоговое меню, с помощью которого он сможет воспользоваться методами разработанных классов. В меню будет выводиться текущее состояние рассматриваемого класса, обрабатываться попытки пользователя ввести непредусмотренные значения.

Примеры меню приведены на рисунках 2.1 2.2 и 2.3

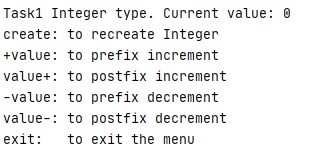
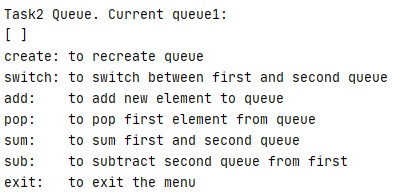
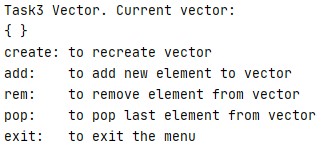
Рис 2.1 меню tInt

Рис 2.2 меню queue



Рис 2.3 меню vector

# **3. Реализация и тестирование программы**

3.1 Описание разработанной программы

*Класс tInt*. Cодержит два конструктора по умолчанию и параметризированный. Первый устанавливает значение счетчика в 0, второй – по заданному. Реализованы префиксные и постфиксные операторы ++ и – увеличивающие и соответственно уменьшающие текущее значение на 1. При выходе за диапазон int, ведет себя соответственно – «перебрасывает» на противоположную границу

*Класс queue*. Содержит конструктор по умолчанию, инициализирующий свои поля. Так как класс шаблонный, может работать с любым поданным типом данных (обозначим T). Реализованы стандартные push(const T&) и pop(), добавляющие новый и соответственно убирающие последний элемент из очереди. Их поведение присвоено операторам += и – соответственно. Реализован оператор=, присваивающий текущему объекту queue переданный. Реализована возможность складывания очередей через оператор+, добавляющий вторую очередь в конец первой, оператор- вычитает элементы второй очереди из первой, если не находит какой-либо – выбрасывает ошибку. Для понимания текущего состояния очереди, перегружен оператор вывода очереди в поток.

*Класс vector*. Содержит 4 конструктора. Конструктор по умолчанию устанавливает текущий размер вектора в 0. Конструктор с одним параметром предопределяет память под заданное количество элементов. Конструктор с параметрами long\* и size\_t копирует переданный массив с его размером. Конструктор копирования присваивает переданный вектор текущему. Реализованы методы копирования массива и переопределения текущей выделенной памяти. Так как для каждого нового элемента переопределение памяти невыгодно, память переопределяется с «запасом» в length\_step(= 10). Реализованы стандартные способы добавления в конец и извлечения элемента – push\_back(const long&) и pop(). Операторы +( и +=) и -(и -=) добавляют и соответственно извлекают определенный элемент, при отсутствии вычитаемого элемента, выбрасывается ошибка. Оператор \*(и \*=) умножает элементы вектора на число. Реализованы префиксные и постфиксные операторы ++ и – добавляющие нулевой и соответственно убирающий последний элемент. Реализована возможность обращения к элементу по индексу. Для понимания текущего состояния вектора, перегружен оператор вывода очереди в поток. Реализована возможность введения вектора из потока в формате: размер, элементы.

*Для пользовательского интерфейса* методы, способные выбросить ошибку (будь то методы классов или ошибки преобразования строки в число), вложены в блоки try catch.

3.2 Тестирование программы

Объект тестирования – классы tInt, queue, vector

Цель тестирования – проверить работоспособность разработанных классов

Средства испытаний – Среда CLion, компилятор MinGW, стандарты языка - 17

Порядок испытаний – по очереди протестированы классы tInt, queue и vector

Методы испытаний – функциональное тестирование (тестирование по входу)

Тестирование проводилось на нескольких наборах данных – в нормальных, граничных и исключительных условиях. Результаты тестирования приведены в таблицах 3.1, 3.2, 3.3:

Табл 3.1 тесты tInt

|  |  |  |
| --- | --- | --- |
| **№ теста** | **Смысл теста** | **Результат** |
| **1** | Проверка работоспособности разработанных функций | Task1 Integer type. Current value: 0  *value+*  Task1 Integer type. Current value: 1  *+value*  Task1 Integer type. Current value: 2  *value-*  Task1 Integer type. Current value: 1  *-value*  Task1 Integer type. Current value: 0  *create 10*  Task1 Integer type. Current value: 10 |
| **2** | Выход за верхний предел | Task1 Integer type. Current value: 2147483647  *value+*  Task1 Integer type. Current value: -2147483648 |
| **3** | Выход за нижний предел | Task1 Integer type. Current value: -2147483648  *-value*  Task1 Integer type. Current value: 2147483647 |
| **4** | Ввод несуществующих команд | *fdsfdsf*  Unknown command  *create ghjk*  Wrong argument. Value set to 0 |

Табл 3.2 тесты queue

|  |  |  |
| --- | --- | --- |
| **№ теста** | **Смысл теста** | **Результат** |
| **1** | Проверка работоспособности разработанных функций | Task2 Queue. Current queue1:  [ ]  *add*  Type value to add  *10*  Task2 Queue. Current queue1:  [ 10 ]  *switch*  Task2 Queue. Current queue2:  [ ]  *add*  Type value to add  *15*  Task2 Queue. Current queue2:  [ 15 ]  *sum*  Task2 Queue. Current queue2:  [ 10 15 ]  *add*  Type value to add  *45*  Task2 Queue. Current queue2:  [ 45 10 15 ]  *switch*  Task2 Queue. Current queue1:  [ 10 ]  *add*  Type value to add  *45*  Task2 Queue. Current queue1:  [ 45 10 ]  *switch*  Task2 Queue. Current queue2:  [ 45 10 15 ]  *sub*  Task2 Queue. Current queue2:  [ 15 ]  *pop*  Task2 Queue. Current queue2:  [ ] |
| **2** | Сложение и вычитание пустых очередей | Task2 Queue. Current queue1:  [ ]  Current queue2:  [ ]  *sum*  Task2 Queue. Current queue1:  [ ]  Current queue2:  [ ]  *sub*  Task2 Queue. Current queue1:  [ ]  Current queue2:  [ ] |
| **3** | Вычитание несуществующих элементов | Task2 Queue. Current queue1:  [ ]  *add*  Type value to add  *15*  Task2 Queue. Current queue1:  [ 15 ]  *switch*  Task2 Queue. Current queue2:  [ ]  *sub*  Unable find element in first queue |
| **4** | Ввод несуществующих команд | Task2 Queue. Current queue1:  [ ]  *sdofk* |
|  |  | Табл 3.2 продолжение |
|  |  | Unknown command  *add*  Type value to add  *lot*  Wrong argument |

Табл 3.3 тесты vector

|  |  |  |
| --- | --- | --- |
| **№ теста** | **Смысл теста** | **Результат** |
| **1** | Проверка работоспособности разработанных функций | Task3 Vector. Current vector:  { }  *add*  Would you like to specify element? y/n  *n*  Task3 Vector. Current vector:  { 0 }  *add*  Would you like to specify element? y/n  *y*  Type element  *15*  Task3 Vector. Current vector:  { 0 15 }  *rem*  Type element to remove  *15*  Task3 Vector. Current vector:  { 0 }  *pop*  Task3 Vector. Current vector:  { }  *create*  How to create vector?  1 - empty vector  2 - with one element  3 - with preallocated memory  4 - type all elements  *1*  Task3 Vector. Current vector:  { }  *create*  How to create vector?  1 - empty vector  2 - with one element  3 - with preallocated memory  4 - type all elements  *2*  Specify element  *92*  Task3 Vector. Current vector:  { 92 }  *create*  How to create vector?  1 - empty vector  2 - with one element  3 - with preallocated memory  4 - type all elements  *3*  Specify count of elements for allocating memory  *12*  Task3 Vector. Current vector:  { }  *create*  How to create vector?  1 - empty vector  2 - with one element  3 - with preallocated memory  4 - type all elements  *4*  Firstly specify count of elements, then type elements  *10*  *0 1 2 3 4 5 6 7 8 9*  Task3 Vector. Current vector:  { 0 1 2 3 4 5 6 7 8 9 } |
| **2** | Попытка удаления несуществующих элементов | Task3 Vector. Current vector:  { }  *add*  Would you like to specify element? y/n |
|  |  | Табл 3.3 продолжение |
|  |  | *y*  Type element  *20*  Task3 Vector. Current vector:  { 20 }  *rem*  Type element to remove  *10*  vector::unable\_to\_find\_element  Task3 Vector. Current vector:  { 20 }  *pop*  Task3 Vector. Current vector:  { }  *pop*  vector::unable\_to\_find\_element  Task3 Vector. Current vector:  { } |

# **Вывод**

Результаты тестов показали, что программа работает, как и задумано.

Научились перегрузке операторов, познакомились с дружескими функциями.

# **Приложение. Код программы**

**main.cpp**

*//Лабораторная работа 4. Черкасов А. Вариант 1. А-08-19.*

#include **<iostream>**#include **"lab\_ui.hpp"  
  
int** main() {  
 lab4Ui ui(std::cout,std::cin);  
 ui.set\_tInt\_ui();  
 ui.show\_ui();  
 ui.set\_queue\_ui();  
 ui.show\_ui();  
 ui.set\_vector\_ui();  
 ui.show\_ui();  
 **return** 0;  
}

**lab4Ui.hpp**

#include **"tInt.h"**#include **"queue.hpp"**#include **"vector.h"**#include **<istream>  
  
class** lab4Ui{  
 std::ostream& os;  
 std::istream& is;  
  
 tInt\* newInt;  
 **bool** is\_first\_queue;  
 queue<**long**>\* firstQueue;  
 queue<**long**>\* secondQueue;  
 vector\* newVector;  
 **unsigned short** type;  
  
**public**:  
 lab4Ui();  
 lab4Ui(std::ostream&, std::istream&);  
 ~lab4Ui();  
 *//Методы подготовки меню под определенный тип данных* **void** set\_tInt\_ui();  
 **void** set\_queue\_ui();  
 **void** set\_vector\_ui();  
 *//Вывод меню* **void** show\_ui();  
  
**private**:  
 *//Обработчики команд* **bool** tInt\_eventHandler(**const int**&);  
 **bool** queue\_eventHandler(**const int**&);  
 **bool** vector\_eventHandler(**const int**&);  
 *//Вывод меню и получение команд от пользователя* **int** tInt\_get\_user\_command();  
 **int** queue\_get\_user\_command();  
 **int** vector\_get\_user\_command();  
};  
  
lab4Ui::lab4Ui(): os(std::cout), is(std::cin), newInt(**nullptr**),  
 firstQueue(**nullptr**), secondQueue(**nullptr**), newVector(**nullptr**), type(0), is\_first\_queue(**false**) {  
}  
  
lab4Ui::lab4Ui(std::ostream& \_os, std::istream& \_is): os(\_os), is(\_is), newInt(**nullptr**),  
 firstQueue(**nullptr**), secondQueue(**nullptr**), newVector(**nullptr**), type(0), is\_first\_queue(**false**) {  
}  
  
*//Все меню работают одинаково: обработчик событий выдает завершилась программа или нет,  
//а метод получения комманд возвращает их код, что позволяет обернуть их в виде  
//цикл с предусловием(обработчик комманд(получение комманды()))***void** lab4Ui::show\_ui() {  
 **bool** show\_ui = **true**;  
 **switch**(type) {  
 **default**:  
 **while** (show\_ui){  
 **try**{  
 show\_ui = tInt\_eventHandler(tInt\_get\_user\_command());  
 } **catch** (std::exception& error) {  
 os<< error.what() << std::endl;  
 }  
 }  
 **delete** newInt;  
 **break**;  
 **case**(1):  
 **while**(show\_ui){  
 **try**{  
 show\_ui = queue\_eventHandler(queue\_get\_user\_command());  
 } **catch** (std::exception& error) {  
 os<< error.what() << std::endl;  
 }  
 }  
 **delete** firstQueue;  
 **delete** secondQueue;  
 **break**;  
 **case**(2):  
 **while**(show\_ui){  
 **try**{  
 show\_ui = vector\_eventHandler(vector\_get\_user\_command());  
 } **catch** (std::exception& error) {  
 os<< error.what() << std::endl;  
 }  
 }  
 **delete** newVector;  
 **break**;  
 }  
}  
  
**void** lab4Ui::set\_tInt\_ui() {  
 newInt = **new** tInt;  
 type = 0;  
}  
  
**void** lab4Ui::set\_queue\_ui() {  
 firstQueue = **new** queue<**long**>;  
 secondQueue = **new** queue<**long**>;  
 is\_first\_queue=**true**;  
 type = 1;  
}  
  
**void** lab4Ui::set\_vector\_ui() {  
 newVector = **new** vector;  
 type=2;  
}  
  
**int** lab4Ui::tInt\_get\_user\_command() {  
 os << **"Task1 Integer type. Current value: "** << (\*newInt) << std::endl <<  
 **"create: to recreate Integer"**<< std::endl <<  
 **"+value: to prefix increment"**<< std::endl <<  
 **"value+: to postfix increment"**<< std::endl <<  
 **"-value: to prefix decrement"**<< std::endl <<  
 **"value-: to postfix decrement"**<< std::endl<<  
 **"exit: to exit the menu"**<< std::endl;  
  
 **int** command\_code=0;  
 std::string command;  
 is>>command;  
  
 **if** (command==**"create"**) command\_code=1;  
 **else if** (command==**"+value"**) command\_code=2;  
 **else if** (command==**"value+"**) command\_code=3;  
 **else if** (command==**"-value"**) command\_code=4;  
 **else if** (command==**"value-"**) command\_code=5;  
 **else if** (command==**"exit"**) command\_code=6;  
  
 **return** command\_code;  
}  
  
  
**bool** lab4Ui::tInt\_eventHandler(**const int**& command) {  
 **switch** (command) {  
 **default**:  
 os << **"Unknown command"** << std::endl;  
 **break**;  
 **case** (1): {  
 std::string buffer;  
 **int** startValue = 0;  
  
 **try** {  
 is >> buffer;  
 startValue = std::stoi(buffer);  
  
 }  
 **catch** (**const** std::exception &) {  
 os << **"Wrong argument. Value set to 0"** << std::endl;  
 }  
 **delete** newInt;  
 newInt = **new** tInt(startValue);  
 }  
 **break**;  
 **case** (2):  
 ++(\*newInt);  
 **break**;  
 **case** (3):  
 (\*newInt)++;  
 **break**;  
 **case** (4):  
 --(\*newInt);  
 **break**;  
 **case** (5):  
 (\*newInt)--;  
 **break**;  
 **case** (6):  
 **return false**;  
 }  
 **return true**;  
}  
  
lab4Ui::~lab4Ui() {  
 **delete** newInt;  
 **delete** firstQueue;  
 **delete** secondQueue;  
 **delete** newVector;  
}  
  
**int** lab4Ui::queue\_get\_user\_command() {  
 queue<**long**>\* current\_queue = (is\_first\_queue) ? firstQueue : secondQueue;  
 os << **"Task2 Queue. Current queue"** << **static\_cast**<**int**>(!is\_first\_queue) + 1 << **":\n"** << (\*current\_queue) << std::endl <<  
 **"create: to recreate queue"** << std::endl <<  
 **"switch: to switch between first and second queue"**<< std::endl <<  
 **"add: to add new element to queue"**<< std::endl <<  
 **"pop: to pop first element from queue"**<< std::endl <<  
 **"sum: to sum first and second queue"**<< std::endl <<  
 **"sub: to subtract second queue from first"**<< std::endl<<  
 **"exit: to exit the menu"**<< std::endl;  
  
 **int** command\_code=0;  
 std::string command;  
 is>>command;  
  
 **if** (command==**"create"**) command\_code=1;  
 **else if** (command==**"add"**) command\_code=2;  
 **else if** (command==**"pop"**) command\_code=3;  
 **else if** (command==**"sum"**) command\_code=4;  
 **else if** (command==**"sub"**) command\_code=5;  
 **else if** (command==**"exit"**) command\_code=6;  
 **else if** (command==**"switch"**) command\_code=7;  
  
 **return** command\_code;  
}  
  
**bool** lab4Ui::queue\_eventHandler(**const int**& command) {  
 **switch** (command) {  
 **default**:  
 os << **"Unknown command"** << std::endl;  
 **break**;  
 **case** (1): {  
 **if** (is\_first\_queue){  
 **delete** firstQueue;  
 firstQueue = **new** queue<**long**>;  
 }  
 **else**{  
 **delete** secondQueue;  
 secondQueue = **new** queue<**long**>;  
 }  
 }  
 **break**;  
 **case** (2): {  
 std::string buffer;  
 os<<**"Type value to add"** << std::endl;  
 **try** {  
 is >> buffer;  
 (\*((is\_first\_queue) ? firstQueue : secondQueue))+=(std::stol(buffer));  
 }  
 **catch** (**const** std::exception &) {  
 os << **"Wrong argument"** << std::endl;  
 }  
 }  
 **break**;  
 **case** (3):  
 --(\*((is\_first\_queue) ? firstQueue : secondQueue));  
 **break**;  
 **case** (4):  
 {  
 queue<**long**> new\_stack = (\*firstQueue);  
 (\*((is\_first\_queue) ? firstQueue : secondQueue)) = new\_stack;  
 }  
 **break**;  
 **case** (5):  
 **try** {  
 (\*((is\_first\_queue) ? firstQueue : secondQueue)) = ((\*firstQueue) - (\*secondQueue));  
 }  
 **catch** (**const** std::exception& error){  
 os << error.what() << std::endl;  
 }  
 **break**;  
 **case** (6):  
 **return false**;  
 **case**(7):  
 is\_first\_queue=!is\_first\_queue;  
 **break**;  
 }  
 **return true**;  
}  
  
**int** lab4Ui::vector\_get\_user\_command() {  
 os << **"Task3 Vector. Current vector:\n"** << (\*newVector) << std::endl <<  
 **"create: to recreate vector"** << std::endl <<  
 **"add: to add new element to vector"**<< std::endl <<  
 **"rem: to remove element from vector"**<< std::endl <<  
 **"pop: to pop last element from vector"**<< std::endl <<  
 **"exit: to exit the menu"**<< std::endl;  
  
 **int** command\_code=0;  
 std::string command;  
 is>>command;  
  
 **if** (command==**"create"**) command\_code=1;  
 **else if** (command==**"add"**) command\_code=2;  
 **else if** (command==**"pop"**) command\_code=3;  
 **else if** (command==**"rem"**) command\_code=4;  
 **else if** (command==**"exit"**) command\_code=5;  
  
 **return** command\_code;  
}  
  
**bool** lab4Ui::vector\_eventHandler(**const int**& command) {  
 **switch** (command) {  
 **default**:  
 os << **"Unknown command"** << std::endl;  
 **break**;  
 **case** (1): {  
 **delete** newVector;  
 **try**{  
 os << **"How to create vector?\n1 - empty vector\n2 - with one element\n3 - with preallocated memory\n4 - type all elements"** << std::endl;  
 **char** ctype;  
 is>>ctype;  
 **switch** (ctype-**'0'**) {  
 **default**:  
 **throw** std::invalid\_argument(**"Unknown command"**);  
 **case**(1):  
 newVector = **new** vector;  
 **break**;  
 **case**(2): {  
 os << **"Specify element"** << std::endl;  
 std::string buffer;  
 **long** first\_element;  
 is >> buffer;  
 first\_element = std::stol(buffer);  
 newVector = **new** vector(first\_element);  
 }  
 **break**;  
 **case**(3): {  
 os << **"Specify count of elements for allocating memory"** << std::endl;  
 std::string buffer;  
 size\_t elements\_count;  
 is >> buffer;  
 elements\_count = std::stoul(buffer);  
 newVector = **new** vector(elements\_count);  
 }  
 **break**;  
 **case**(4):{  
 newVector = **new** vector;  
 os << **"Firstly specify count of elements, then type elements themselves"** << std::endl;  
 is >> (\*newVector);  
 }  
 **break**;  
 }  
 } **catch** (**const** std::exception& error) {  
 os << error.what() << std::endl << **"Created standart vector"** << std::endl;  
 newVector = **new** vector;  
 }  
 }  
 **break**;  
 **case** (2): {  
 os << **"Would you like to specify element? y/n"** << std::endl;  
 **char** command\_char;  
 is>>command\_char;  
 **switch** (command\_char) {  
 **case**(**'y'**): {  
 os << **"Type element"** << std::endl;  
 **long** element=0;  
 **try** {  
 std::string buffer;  
 is >> buffer;  
 element = std::stol(buffer);  
 }  
 **catch** (**const** std::exception &) {  
 os << **"Wrong argument. Value set to zero"**<<std::endl;  
 }  
 (\*newVector)+=element;  
 }  
 **break**;  
 **case**(**'n'**):  
 (\*newVector)++;  
 **break**;  
 **default**:  
 os << **"Unknown command"** << std::endl;  
 **break**;  
 }  
 }  
 **break**;  
 **case** (3):  
 --(\*newVector);  
 **break**;  
 **case** (4):  
 {  
 os << **"Type element to remove"** << std::endl;  
 **long** element=0;  
 **while**(!element) {  
 **try** {  
 std::string buffer;  
 is >> buffer;  
 element = std::stol(buffer);  
 }  
 **catch** (**const** std::exception &) {  
 os << **"Wrong argument. Type again"** << std::endl;  
 }  
 }  
 **try**{  
 (\*newVector)-=element;  
 }**catch** (**const** std::exception& error) {  
 os << error.what() << std::endl;  
 }  
 }  
 **break**;  
 **case** (5):  
 **return false**;  
 }  
 **return true**;  
}

**tInt.h**

#ifndef **LAB4\_TINT\_H**#define **LAB4\_TINT\_H**#include **<istream>  
  
class** tInt {  
 *//Текущее значение* **int**\* value;  
**public**:  
 *//Конструкторы* tInt();  
 **explicit** tInt(**const int**&);  
 *//Prefix* tInt& **operator**++();  
 *//Postfix* tInt **operator**++(**int**);  
 *//Prefix* **friend** tInt& **operator**--(tInt&);  
 *//Postfix* **friend** tInt **operator**--(tInt&,**int**);  
  
 **friend** std::ostream& **operator**<<(std::ostream&,**const** tInt&);  
  
 ~tInt();  
};  
  
  
#endif *//LAB4\_TINT\_H*

**tInt.cpp**

#include **"tInt.h"**#include **<malloc.h>**tInt::tInt() {  
 value = (**int**\*)malloc(**sizeof**(**int**));  
 \*value = 0;  
}  
  
tInt::tInt(**const int**& start\_value) {  
 value = (**int**\*)malloc(**sizeof**(**int**));  
 \*value = start\_value;  
}  
  
tInt& tInt::**operator**++() {  
 ++(\*value);  
 **return** \***this**;  
}  
  
tInt tInt::**operator**++(**int**) {  
 tInt temp(\***this**->value);  
 (\*value)++;  
 **return** temp;  
}  
  
tInt &**operator**--(tInt& in) {  
 --(\*in.value);  
 **return** in;  
}  
  
tInt **operator**--(tInt& in, **int**) {  
 tInt temp(\*in.value);  
 (\*in.value)--;  
 **return** temp;  
}  
  
tInt::~tInt() {  
 free(value);  
}  
  
std::ostream& **operator**<<(std::ostream& os,**const** tInt& out) {  
 os << \*out.value;  
 **return** os;  
}

**queue.hpp**

#ifndef **LAB4\_QUEUE\_HPP**#define **LAB4\_QUEUE\_HPP**#include **<malloc.h>**#include **<istream>**#include **<vector>**#include **<algorithm>  
  
template**<**class** T>  
**class** queue {  
 std::vector<T> array;  
**public**:  
 queue();  
 ~queue();  
 *//Standart push and pop* **void** push(**const** T&);  
 T pop();  
 *//Overloaded+= acting as push* queue& **operator**+=(**const** T&);  
 *//Prefix and postfix -- acting as pop* queue& **operator**--();  
 queue **operator**--(**int**);  
 *//Сложение и вычитание очередей* queue **operator**+(**const** queue&);  
 queue **operator**-(**const** queue&);  
  
 queue& **operator**=(**const** queue&);  
 *//Вывод в поток* **template**<**class** T1>  
 **friend** std::ostream& **operator**<<(std::ostream&,**const** queue<T1>&);  
  
};  
  
**template**<**class** T>  
queue<T>::~queue() = **default**;  
  
**template**<**class** T>  
queue<T>::queue() = **default**;  
  
**template**<**class** T>  
**void** queue<T>::push(**const** T& item) {  
 array.insert(array.begin(),item);  
}  
  
**template**<**class** T>  
T queue<T>::pop() {  
 T item = array[array.size()-1];  
 array.pop\_back();  
 **return** item;  
}  
  
**template**<**class** T>  
queue<T>& queue<T>::**operator**+=(**const** T& item) {  
 push(item);  
 **return** \***this**;  
}  
  
**template**<**class** T>  
queue<T>& queue<T>::**operator**=(**const** queue& in) {  
 **this**->array=in.array;  
 **return** \***this**;  
}  
  
**template**<**class** T>  
queue<T>& queue<T>::**operator**--() {  
 pop();  
 **return** \***this**;  
}  
  
**template**<**class** T>  
queue<T> queue<T>::**operator**--(**int**) {  
 queue<T> prev\_stack = (\***this**);  
 pop();  
 **return** prev\_stack;  
}  
  
**template**<**class** T>  
queue<T> queue<T>::**operator**-(**const** queue& decr) {  
 queue<T> new\_stack = \***this**;  
 **for** (**const** T& item : decr.array){  
 **auto** item\_position = std::find(new\_stack.array.begin(),new\_stack.array.end(),item);  
 **if** (item\_position==new\_stack.array.end()) **throw** std::invalid\_argument(**"Unable find element in first queue"**);  
 new\_stack.array.erase(item\_position);  
 }  
 **return** new\_stack;  
}  
  
**template**<**class** T>  
queue<T> queue<T>::**operator**+(**const** queue& incr) {  
 queue<T> new\_stack = \***this**;  
 **for** (**auto** item\_iter = incr.array.rbegin(); item\_iter<incr.array.rend();item\_iter++)  
 new\_stack.push(\*item\_iter);  
 **return** new\_stack;  
}  
  
**template**<**class** T>  
std::ostream &**operator**<<(std::ostream& os, **const** queue<T>& out) {  
 os << **"[ "**;  
 **for** (**const** T& item: out.array) os << item << **" "**;  
 os << **"]"**;  
 **return** os;  
}  
  
  
#endif *//LAB4\_QUEUE\_HPP*

**vector.h**

#ifndef **LAB4\_VECTOR\_H**#define **LAB4\_VECTOR\_H**#include **<istream>  
  
class** vector {  
 *//Массив, размер, количество элементов* **long**\* array;  
 size\_t length;  
 size\_t element\_count;  
  
 **static const** size\_t length\_step = 10;  
 *//Ошибка на случай отстутствия элемента* **static** std::invalid\_argument no\_element;  
**public**:  
 vector();  
 **explicit** vector(**const** size\_t&);  
 vector(**long**\*,**const** size\_t&);  
 vector(**const** vector&);  
 ~vector();  
 *//Простые push\_back и pop* **void** push\_back(**const long**&);  
 **long** pop();  
 *//Добавление, вычитаение и умножение элемнта(ов)* **friend** vector **operator**+(**const** vector&,**const long**&);  
 **friend** vector **operator**-(**const** vector&,**const long**&);  
 **friend** vector **operator**\*(**const** vector&,**const long**&);  
  
 vector& **operator**=(**const long**&);  
 *//Добавление, вычитаение и умножение элемнта(ов)* vector& **operator**+=(**const long**&);  
 vector& **operator**-=(**const long**&);  
 vector& **operator**\*=(**const long**&);  
 *//Индексация* **long**& **operator**[](**const** size\_t&);  
 *//Префиксное и постфиксное добавление нулевого элемента* vector& **operator**++();  
 vector **operator**++(**int**);  
 *//Префиксное и постфиксное pop* vector& **operator**--();  
 vector **operator**--(**int**);  
  
 **friend** std::ostream& **operator**<<(std::ostream&,**const** vector&);  
 **friend** std::istream& **operator**>>(std::istream&,vector&);  
  
 vector& **operator**=(**const** vector&);  
**private**:  
 **void** reallocate\_memory(**const** size\_t&);  
  
 **static void** copy(**const long**\*,**const** size\_t&,**long**\*);  
};  
  
#endif *//LAB4\_VECTOR\_H*

**vector.cpp**

#include **"vector.h"**#include **<malloc.h>**std::invalid\_argument vector::no\_element = std::invalid\_argument(**"vector::unable\_to\_find\_element"**);  
  
vector::vector():array(**nullptr**),length(0),element\_count(0) {  
}  
  
vector::vector(**const** size\_t& new\_size):array(**nullptr**),length(0),element\_count(0) {  
 reallocate\_memory(new\_size);  
}  
  
**void** vector::reallocate\_memory(**const** size\_t& new\_size) {  
 **long**\* new\_array = (**long**\*)malloc(**sizeof**(**long**)\*new\_size);  
 copy(array,element\_count,new\_array);  
 **for**(size\_t i=element\_count;i<new\_size;i++) new\_array[i] = **long**();  
 free(array);  
 array=new\_array;  
 length=new\_size;  
}  
  
**void** vector::copy(**const long**\* source,**const** size\_t& source\_size,**long**\* new\_array) {  
 **for** (size\_t i=0;i<source\_size;i++){  
 new\_array[i]=source[i];  
 }  
}  
  
  
vector::vector(**long**\* new\_array, **const** size\_t& new\_size) {  
 reallocate\_memory(new\_size+length\_step);  
 copy(new\_array,new\_size,array);  
 element\_count=new\_size;  
}  
  
vector::vector(**const** vector& source):array(**nullptr**),length(0),element\_count(0) {  
 \***this** = source;  
}  
  
vector &vector::**operator**=(**const** vector& source) {  
 **if** (source.array!=**this**->array) {  
 free(array);  
 array = (**long** \*) malloc(source.length \* **sizeof**(**long**));  
 copy(source.array, source.element\_count, **this**->array);  
 **for** (size\_t i = source.element\_count; i < source.length; i++) **this**->array[i] = **long**();  
 **this**->element\_count = source.element\_count;  
 **this**->length = source.length;  
 }  
 **return** \***this**;  
}  
  
vector::~vector() {  
 free(array);  
}  
  
**void** vector::push\_back(**const long**& new\_element) {  
 **if** (element\_count==length)  
 reallocate\_memory(length + length\_step);  
 array[element\_count]=new\_element;  
 element\_count++;  
}  
  
**long** vector::pop() {  
 **if** (element\_count) {  
 **long** value = array[--element\_count];  
 **if** (element\_count + length\_step == length)  
 reallocate\_memory(element\_count);  
 **return** value;  
 }  
 **throw** no\_element;  
}  
  
**long** &vector::**operator**[](**const** size\_t& position) {  
 **if** (position>=length) {  
 reallocate\_memory(position+length\_step);  
 element\_count=position;  
 }  
 **else if** (position>=element\_count) element\_count=position;  
 **return** array[position];  
}  
  
vector& vector::**operator**=(**const long**& element) {  
 \***this** = vector();  
 push\_back(element);  
 **return** \***this**;  
}  
  
std::ostream &**operator**<<(std::ostream& os,**const** vector& out) {  
 os << **"{ "**;  
 **for** (size\_t i=0;i<out.element\_count;i++) os << out.array[i] << **" "**;  
 os << **"}"**;  
 **return** os;  
}  
  
vector &vector::**operator**+=(**const long**& new\_element) {  
 push\_back(new\_element);  
 **return** \***this**;  
}  
  
vector& vector::**operator**-=(**const long**& decr\_element) {  
 size\_t index = element\_count;  
 **for** (size\_t i=0; i < element\_count && index == element\_count; i++)  
 **if** (array[i] == decr\_element) index=i;  
 **if** (index == element\_count) **throw** vector::no\_element;  
  
 **for** (size\_t i=index;i+1 < element\_count; i++)  
 array[i]=array[i + 1];  
 array[--element\_count] = **long**();  
 **return** \***this**;  
}  
  
vector **operator**-(**const** vector& in, **const long**& decr\_element) {  
 vector mod\_vector = in;  
 mod\_vector-=decr\_element;  
 **return** mod\_vector;  
}  
  
vector **operator**+(**const** vector& in, **const long**& new\_element) {  
 vector mod\_vector = in;  
  
 mod\_vector+=new\_element;  
 **return** mod\_vector;  
}  
  
vector **operator**\*(**const** vector& in, **const long**& coeff) {  
 vector mod\_vector = in;  
 mod\_vector\*=coeff;  
 **return** mod\_vector;  
}  
  
vector& vector::**operator**\*=(**const long**& coeff) {  
 **for** (size\_t i=0;i<element\_count;i++) (\***this**)[i]\*=coeff;  
 **return** \***this**;  
}  
  
vector vector::**operator**--(**int**) {  
 vector temp = \***this**;  
 pop();  
 **return** temp;  
}  
  
vector& vector::**operator**--() {  
 pop();  
 **return** \***this**;  
}  
  
vector &vector::**operator**++() {  
 push\_back(**long**());  
 **return** \***this**;  
}  
  
vector vector::**operator**++(**int**) {  
 vector temp = \***this**;  
 push\_back(**long**());  
 **return** temp;  
}  
  
std::istream &**operator**>>(std::istream& is, vector& new\_vector) {  
 std::string buffer;  
 size\_t elements\_count;  
 is >> buffer;  
 elements\_count = std::stoul(buffer);  
 new\_vector = vector(elements\_count + vector::length\_step);  
 **for** (size\_t i = 0; i < elements\_count; i++) {  
 is >> buffer;  
 new\_vector += std::stol(buffer);  
 }  
 **return** is;  
}