Дисциплина: Объектно-ориентированное программирование

Лабораторная работа №1 «Реализация контейнерного класса»

Задание: разработать шаблонный контейнерный класс в соответствии с вариантом, а также класс итератора к нему. Протестировать разработанный класс. Для каждого варианта указан список публичных методов и пояснения к ним – это минимальный набор методов, которые необходимо реализовать для данного варианта. Если операция не может быть выполнена, необходимо выбросить исключение.

Студентом по желанию могут быть реализованы и другие методы, которые он сочтет полезными и применимыми для данного класса.

При разработке следует руководствоваться принципами ООП. Использование контейнеров из библиотек запрещено. В рамках лабораторной работы необходимо использовать следующие паттерны: свойства (properties), итератор (iterator).

Тестирование

Тестирование производить при помощи тестовой программы (достаточно консольной). Проверить необходимо все публичные методы реализованного класса.

Бонусные задания

1. Возможно использование специальной библиотеки для тестирования (в этом случае ожидается базовое понимание механизмов работы библиотеки тестирования, используемых в ней макросов).
2. Реализация дополнительных методов, входящих в состав библиотек std и Qt.

Класс итератора

Класс, необходимый для реализации всех вариантов:

Метод	Пояснение
Iterator<T>(тип_контейнера_из_варианта <T> container_obj)	конструктор, принимающий объект контейнерного класса, который необходимо обойти с помощью данного итератора
Iterator<T> next()	перейти к следующему объекту в контейнере
T value()	получить значение текущего объекта в контейнере
bool is_end()	указывает ли итератор на конечный фиктивный элемент контейнера, следующий

	за последним реальным. Нужен для определения конца итерирования
Iterator<T> &operator++()	префиксный инкремент, эквивалентен next()
T &operator*()	оператор разыменования, эквивалентен value()
bool operator ==(Iterator<T> &b)	оператор сравнения
bool operator !=(Iterator<T> &b)	оператор сравнения

Задания по вариантам

Вариант определяется как остаток от деления номера студента в группе на 4.

Вариант 0 Список

Метод	Пояснение
list();	конструктор по умолчанию
list(const list<T>& lst);	конструктор копирования
list(list<T>&& list);	конструктор перемещения
explicit list(std::initializer_list<T> lst);	явный конструктор со списком инициализации
~list();	деструктор
list<T>& operator =(const list<T>& lst);	перегрузка оператора присваивания
int get_length() const;	получить текущий размер списка
void add(const T& elem);	добавить элемент в конец списка
void add_range(const list<T>& lst);	добавить список элементов в конец списка
void add_range(T* arr, int size);	добавить массив элементов в конец списка
void set_elem(int index,const T& elem);	изменить элемент списка по индексу
T& get_elem(int index);	получить элемент списка по индексу
void remove_elem(int index);	удалить элемент списка по индексу
list<T> combine(const list<T>& lst);	объединение списка с другим списком (метод возвращает новый список, содержащий сначала элементы текущего списка, затем, переданного в Combine)
void sort(int (*comp)(const T& r1, constT& r2));	отсортировать список используя переданный компаратор
int get_index(T &elem) const;	если содержится в списке элемент T, возвращает индекс элемента или -1 в случае, если элемент не найден
T* to_array();	создать новый массив, в который необходимо

	записать все элементы вектора
T& operator[](int index);	доступ к элементу аналогично массиву

Вариант 1 Матрица

Метод	Пояснение
matrix(unsigned int n, unsigned int m);	конструктор, создающий пустую матрицу заданного размера
matrix(const matrix<T>& matr);	конструктор копирования
matrix(matrix<T>&& matr);	конструктор перемещения
explicit matrix(std::initializer_list<std::initializer_list< T>> lst);	конструктор со списком инициализации
~matrix();	деструктор
matrix<T>& operator =(const matrix<T>& matr);	перегрузка оператора присваивания для двух матриц типа T
matrix<T>& operator +=(const matrix<T>& matr);	перегрузка оператора += для двух матриц типа T
matrix<T>& operator -=(const matrix<T>& matr);	перегрузка оператора -= для двух матриц типа T
template<typename _T> friend matrix<_T> operator +(const matrix<_T>& m1, const matrix<_T>& m2);	перегрузка оператора + для двух матриц
template<typename _T> friend matrix<_T> operator -(const matrix<_T>& m1, const matrix<_T>& m2);	перегрузка оператора - для двух матриц
template<typename _T> friend matrix<_T> operator *(const matrix<_T>& m1, const matrix<_T>& m2);	перегрузка оператора * для двух матриц.
template<typename _T> friend matrix<_T> operator +(const matrix<_T>& m1, double num);	перегрузка оператора + для сложения матрицы с числом
template<typename _T> friend matrix<_T> operator -(const matrix<_T>& m1, double num);	перегрузка оператора – для вычисления разности матрицы с числом

template<typename _T> friend matrix<_T> operator /(const matrix<_T>& m1, double num);	перегрузка оператора / для вычисления частного матрицы с числом
template<typename _T> friend matrix<_T> operator *(const matrix<_T>& m1, double num);	перегрузка оператора * для умножения матрицы с числом
template<typename _T> friend std::ostream& operator <<(std::ostream& os, const matrix<_T>& matr);	перегрузка оператора << для вывода класса в поток (cout к примеру)
void set_elem(unsigned int i, unsigned int j,const T& elem);	метод изменения элемента матрицы по индексу
T& get_elem(unsigned int i, unsigned int j);	метод получения элемента матрицы по индексу
T& operator ()(unsigned int i, unsigned int j);	метод получения элемента матрицы по индексу, через синтаксис круглых скобок. matrix(i, j)
bool is_square();	метод проверки матрицы на квадратную
unsigned int get_row() const;	метод получения числа строк матрицы
unsigned int get_columns() const;	метод получения числа столбцов матрицы
Iterator<T> iterator_begin()	метод получения итератора на начало матрицы (первый элемент).
Iterator<T> iterator_end()	метод получения итератора на конец матрицы (конец – это фиктивный элемент, следующий за последним в матрице)

Вариант 2 Математический вектор

Метод	Пояснение
m_vector(int length);	конструктор с указанием размерности
m_vector(const m_vector<T>& vect);	конструктор копирования
m_vector (m_vector <T>&& vect);	конструктор перемещения
explicit m_vector(std::initializer_list<T> lst);	конструктор со списком инициализации
~m_vector();	деструктор
m_vector<T>& operator =(const m_vector<T>& lst);	перегрузка оператора присваивания

int get_length() const;	получить текущий размер
void set_elem(int index,const T& elem);	изменить элемент вектора по индексу
T& get_elem(int index);	получить элемент списка по индексу
T* to_array();	создать новый массив, в который необходимо записать все элементы вектора
T& operator[](int index);	доступ к элементу, аналогично массиву
template<typename _T> friend std::ostream& operator<<(std::ostream& os, const m_vector<_T>& lst);	перегрузка оператора << для вывода класса в поток (cout к примеру)
m_vector<T>& operator +=(const m_vector<T>& vect);	перегрузка оператора +=, к this добавлется vect
m_vector<T>& operator -=(const m_vector<T>& vect);	перегрузка оператора -=, из this вычитается vect
m_vector<T>& operator *=(const T& val);	перегрузка оператора *=, каждый элемент this домножается на val
m_vector<T>& operator /=(const T& val);	перегрузка оператора /=, каждый элемент this делится на val
template<typename _T> friend m_vector<_T> operator +(const m_vector<_T>& v1, const m_vector<_T>& v2);	перегрузка оператора += к v1 добавлется v2
template<typename _T> friend m_vector<_T> operator -(const m_vector<_T>& v1, const m_vector<_T>& v2);	перегрузка оператора -, из v1 вычитается v2
template<typename _T> friend m_vector<_T> operator *(const m_vector<_T>& v1, const T& val);	перегрузка оператора *, каждый элемент v1 домножается на val
template<typename _T> friend m_vector<_T> operator /(const m_vector<_T>& v1, const T& val);	перегрузка оператора /, каждый элемент v1 делится на val
Iterator<T> iterator_begin()	метод получения итератора на начало вектора (первый элемент)

Iterator<T> iterator_end()	метод получения итератора на конец списка (фиктивный элемент, следующий за последним в векторе)

Вариант 3 Множество (добавляемые элементы уникальны)

Метод	Пояснение
set();	конструктор по умолчанию
set(const set<T>& s);	конструктор копирования
set (set <T>&& s);	конструктор перемещения
explicit set(std::initializer_list<T> lst);	конструктор со списком инициализации
~set();	деструктор
set<T>& operator =(const set<T>& lst);	перегрузка оператора присваивания
int get_length() const;	получить текущий размер
bool contains(const T& elem);	проверить наличие в множестве элемента
void add(const T& elem);	добавить элемент в множество
void remove(const T& elem);	удалить элемент из множества
T* to_array();	создать новый массив, в который необходимо записать все элементы вектора
set<T>& union(const set<T>& s);	результат – объединение this с s
set<T>& intersection(const set<T>& s);	результат – пересечение this с s
set<T>& subtract(const set<T>& s);	результат – разность this и s
template<typename _T> friend std::ostream& operator <<(std::ostream& os, const set<_T>& lst);	перегрузка оператора << для вывода класса в поток (cout к примеру)
set<T>& operator +=(const set<T>& s);	перегрузка оператора += результат – объединение множеств this и s
set<T>& operator *=(const set<T>& s);	перегрузка оператора *=, результат – пересечение множеств this и s
set<T>& operator /=(const set<T>& s);	перегрузка оператора /=, разность множеств this и s
template<typename _T> friend set<_T> operator +(const set<_T>& s1, const set<_T>& s2);	перегрузка оператора + результат – объединение множеств v1 и v2
template<typename _T>	перегрузка оператора *, результат – пересечение

friend set<_T> operator *(const set<_T>& s1, const set<_T>& s2);	множеств v1 и v2
template<typename _T> friend set<_T> operator /(const set<_T>& s1, const set<T>& s2);	перегрузка оператора /, разность множеств v1 и v2
Iterator<T> iterator_begin()	метод получения итератора на начало множества (первый элемент)
Iterator<T> iterator_end()	метод получения итератора на конец множества (фиктивный элемент, следующий за последним в множестве)
void clear();	очистить множество

Бонусный вариант Дерево

Не более 5 человек из группы могут заменить свой вариант на бонусный.

На этот вариант нет четкого описания методов, его требуется составить самостоятельно, после чего утвердить у преподавателя. Не допускается сдача бонусных заданий с одинаковыми методами. Лабораторная будет засчитана первому сдавшему.