ПРАКТИЧЕСКИЕ РАБОТЫ

Практическая работа №1. Работа с последовательными контейнерами

Задание 1.1

Постройте связный список (используйте класс list библиотеки STL), который содержит объекты указанного в таблице 1.1 типа T. Постройте функции добавления push() и удаления pop() элементов таким образом, чтобы список оставался отсортированным при выполнении этих операций (допустимо удаление из начала контейнера, с конца и из произвольного места). Постройте функцию filter(), которая принимает предикат P (см. таблицу 1.1) и возвращает новый список с объектами, для которых предикат принимает истинное значение. Постройте функцию вывода содержимого списка с помощью итераторов.
Примечание: *В этом задании не требуется создавать класс списка, нужно использовать класс list из библиотеки STL и написать отдельно требуемые функции (не методы класса).*

Код 1.1. Пример функции добавления в список элемента с сохранением упорядоченности

```
#include <list>
#include <iostream>

using namespace std;


template<class T>
void insert(list<T>& lst, T element)
{
	list<T>::iterator p = lst.begin();
	while (p != lst.end())
	{
		if (*p > element)
			break;
		p++;
	}
	lst.insert(p, element);
}
```

```
int main()
{
	list<char> lst;
	int i=0;
	for(i=0;i<10;i+=2)
		lst.push_back('A' + i);

	insert(lst, 'X');
	list<char>::iterator p = lst.begin();
	while(p!=lst.end())
	{
		//перемещение по контейнеру с помощью указателя, нет
операции [i]
		cout<<*p<<"";
		p++;
	}
	return 0;
}
```

Таблица 1.1. Варианты типов хранимых в контейнере значений и условие предиката в функции фильтрации

Вариант	Тип T	Условие предиката P
1.	char	Только буквы верхнего регистра. Сортировка по коду символа.
2.	double	Только положительные числа
3.	int	Только простые числа
4.	Complex	Только комплексные числа с отрицательной действительной частью. Сортировка по модулю комплексного числа.
5.	Point2D	Только точки, лежащие во втором октанте. Сортировка по расстоянию до центра координат.
6.	Fraction	Только правильные дроби. Сортировка по величине дроби.
7.	char	Только гласные
8.	double	Числа, модули которых больше некоторого значения *a*
9.	int	Только числа, являющиеся факториалами
10.	int	Только квадраты некоторых целых чисел (1, 4, 9, 16 и

		т.д.)
11.	Complex	Только чисто мнимые числа. Сортировка по модулю комплексного числа.
12.	Fraction	Только дроби с числителями, представляющими простые числа.
13.	Point2D	Только точки, лежащие за пределами единичного круга.
14.	int	Только элементы последовательности Фибоначчи
15.	Fraction	Дроби, по модулю превосходящие некоторое значения *a*
16.	char	Только согласные.
17.	double	Числа, модули которых меньше некоторого значения *a*
18.	int	Только числа, кратные 3
19.	Complex	Только комплексные числа с четной действительной частью. Сортировка по модулю комплексного числа.
20.	Point2D	Только точки, лежащие внутри единичного квадрата с центров в начале координат. Сортировка по расстоянию до центра координат.
21.	Fraction	Только дроби, у которых числитель квадрат некоторого числа. Сортировка по величине дроби.
22.	char	Только буквы нижнего регистра.
23.	double	Числа, дробная часть которых не превосходит *a*
24.	int	Только числа, являющиеся факториалами четных значений
25.	int	Только кубы некоторых целых чисел (1, 8, 27, 64 и т.д.)
26.	Complex	Только комплексные числа с нечетной действительной и мнимой частью. Сортировка по модулю комплексного числа.
27.	Fraction	Только дроби с числителями, представляющими простые числа.
28.	Point2D	Только точки, лежащие внутри единичного круга.
29.	int	Только числа, кратные 7, отрицательные
30.	Fraction	Дроби, по модулю, не превосходящие некоторое значения *a*

Задание 1.2

Заполните список из пункта 1 объектами класса C (таблица 1.2), сохраняя убывание по приоритету: полю или группе полей, указанных в варианте. Функция pop() должна удалять объект из контейнера и возвращать

как результат объект с наибольшим приоритетом (определяется по полям, указанным в третьем столбце таблицы 1.2: больший приоритет имеет объект с большим значением первого поля; если значение первого поля совпадает, то сравниваются значения второго поля и так далее). Если больший приоритет имеют объекты с меньшим значением поля (упорядоченность по возрастанию), это указано в скобках.

Пример из варианта 1: *объекты недвижимости сортируются по убыванию цены. Если цена совпадает, то сравниваем по адресу, но для адреса уже используется упорядочение по возрастанию (“меньший” адрес - больший приоритет, строки сравниваются в лексикографическом порядке, “как в словаре”).*

Таблица 1.2. Варианты типов хранимых в контейнере значений и порядок сравнения полей для упорядочения объектов

Вариант	Класс С	Приоритет
1.	«Объект жилой недвижимости». Минимальный набор полей: адрес, тип (перечислимый тип: городской дом, загородный дом, квартира, дача), общая площадь, жилая площадь, цена.	Цена; адрес (по возрастанию)
2.	«Сериал». Минимальный набор полей: название, продюсер, количество сезонов, популярность, рейтинг, дата запуска, страна.	Рейтинг; название (по возрастанию)
3.	«Смартфон». Минимальный набор полей: название, размер экрана, количество камер, объем аккумулятора, максимальное количество часов без подзарядки, цена.	Цена, количество камер, размер экрана; название марки (по возрастанию)
4.	«Спортсмен». Минимальный набор полей: фамилия, имя, возраст, гражданство, вид спорта, количество медалей.	Количество медалей; возраст (по возрастанию); фамилия и имя (по возрастанию)
5.	«Врач». Минимальный набор полей: фамилия, имя,	Рейтинг, стаж; фамилия и имя (по возрас-

	специальность, должность, стаж, рейтинг (вещественное число от 0 до 100).	танию)
6.	«Авиакомпания». Минимальный набор полей: название, международный код, количество обслуживаемых линий, страна, интернет-адрес сайта, рейтинг надёжности (целое число от -10 до 10).	Надёжность, количество обслуживаемых линий; название (по возрастанию)
7.	«Книга». Минимальный набор полей: фамилия (первого) автора, имя (первого) автора, название, год издания, название издательства, число страниц, вид издания (перечислимый тип: электронное, бумажное или аудио), тираж.	Тираж; год издания (по возрастанию); название (по возрастанию)
8.	«Небесное тело». Минимальный набор полей: тип (перечислимый тип: астероид, естественный спутник, планета, звезда, квазар), имя (может отсутствовать), номер в небесном каталоге, удаление от Земли, расчётная масса в миллиардах тонн (для сверхбольших объектов допускается значение Inf, которое должно корректно обрабатываться).	Масса; номер в каталоге (по возрастанию)
9.	«Населённый пункт». Минимальный набор полей: название, тип (перечислимый тип: город, посёлок, село, деревня), числовой код региона, численность населения, площадь.	Площадь, численность населения; числовой код региона (по возрастанию)
10.	«Музыкальный альбом». Минимальный набор полей: имя или псевдоним исполнителя, название альбома, количество композиций, год выпуска, количество проданных экземпляров.	Количество проданных экземпляров; количество композиций; год выпуска (по возрастанию); имя или псевдоним исполнителя (по возрастанию)
11.	«Фильм».	Доход, стоимость; год

	Минимальный набор полей: фамилия, имя режиссёра, название, страна, год выпуска, стоимость, доход.	выпуска (по возрастанию); фамилия и имя режиссера (по возрастанию); название фильма (по возрастанию)
12.	«Автомобиль». Минимальный набор полей: имя модели, цвет, серийный номер, количество дверей, год выпуска, цена.	Цена; год выпуска; марка (по возрастанию); серийный номер (по возрастанию)
13.	«Автовладелец». Минимальный набор полей: фамилия, имя, регистрационный номер автомобиля, дата рождения, номер техпаспорта.	Регистрационный номер автомобиля; номер техпаспорта; фамилия и имя автовладельца (по возрастанию)
14.	«Стадион». Минимальный набор полей: название, виды спорта, год постройки, вместимость, количество арен.	Вместимость, количество арен, год постройки; название (по возрастанию)
15.	«Спортивная Команда». Минимальный набор полей: название, город, число побед, поражений, ничьих, количество очков.	Число побед, число ничьих; число поражений (по возрастанию); название (по возрастанию)
16.	«Пациент». Минимальный набор полей: фамилия, имя, дата рождения, телефон, адрес, номер карты, группа крови.	Номер карты; группа крови; фамилия и имя (по возрастанию)
17.	«Покупатель». Минимальный набор полей: фамилия, имя, город, улица, номера дома и квартиры, номер счёта, средняя сумма чека.	Средняя сумма чека; номер счёта; фамилия и имя (по возрастанию)
18.	«Школьник». Минимальный набор полей: фамилия, имя,	Класс; дата рождения (по возрастанию); фа-

	пол, класс, дата рождения, адрес.	милия и имя (по возрастанию)
19.	«Человек». Минимальный набор полей: фамилия, имя, пол, рост, возраст, вес, дата рождения, телефон, адрес.	Возраст, рост; вес (по возрастанию); фамилия и имя (по возрастанию)
20.	«Государство». Минимальный набор полей: название, столица, язык, численность населения, площадь.	Численность населения; площадь; название (по возрастанию)
21.	«Сайт». Минимальный набор полей: название, адрес, дата запуска, язык, тип (блог, интернет-магазин и т.п.), cms, дата последнего обновления, количество посетителей в сутки.	Количество посетителей в сутки, дата последнего обновления; адрес (по возрастанию)
22.	«Программа». Минимальный набор полей: название, версия, лицензия, есть ли версия для android, iOS, платная ли, стоимость, разработчик, открытость кода, язык кода.	Стоимость, версия; название (по возрастанию)
23.	«Ноутбук». Минимальный набор полей: производитель, модель, размер экрана, процессор, количество ядер, объем оперативной памяти, объем диска, тип диска, цена.	Цена, количество ядер, объем оперативной памяти, размер экрана; модель (по возрастанию)
24.	«Велосипед». Минимальный набор полей: марка, тип, тип тормозов, количество колес, диаметр колеса, наличие амортизаторов, детский или взрослый.	Диаметр колеса, количество колес; марка (по возрастанию)
25.	«Программист». Минимальный набор полей: фамилия, имя, email, skype, telegram, основной язык программирования, текущее место работы, уровень (число от 1 до 10).	Уровень; основной язык программирования (по возрастанию); фамилия и имя (по возрастанию)
26.	«Профиль в соц.сети».	Количество друзей;

	Минимальный набор полей: псевдоним, адрес страницы, возраст, количество друзей, интересы, любимая цитата.	псевдоним (по возрастанию)
27.	«Супергерой». Минимальный набор полей: псевдоним, настоящее имя, дата рождения, пол, суперсила, слабости, количество побед, рейтинг силы.	Рейтинг силы, количество побед; псевдоним (по возрастанию)
28.	«Фотоаппарат». Минимальный набор полей: производитель, модель, тип, размер матрицы, количество мегапикселей, вес, тип карты памяти, цена.	Цена, вес, размер матрицы; модель (по возрастанию)
29.	«Файл». Минимальный набор полей: полный адрес, краткое имя, дата последнего изменения, дата последнего чтения, дата создания.	Дата последнего изменения, дата последнего чтения; полный адрес (по возрастанию)
30.	«Самолет». Минимальный набор полей: название, производитель, вместимость, дальность полета, максимальная скорость.	Вместимость, дальность полета; производитель (по возрастанию), название (по возрастанию)

Задание 1.3

Постройте шаблон класса двусвязного списка путём наследования от класса IteratedLinkedList. Реализуйте функции добавления элемента push() и удаления элемента pop() в классе-наследнике D (для четных вариантов D – Стек, для нечетных – Очередь) согласно схеме: для класса Стек элементы добавляются в конец, извлекаются с конца; для класса Очередь элементы добавляются в конец, извлекаются с начала. Постройте наследник класса D. Переопределите функцию добавления нового элемента таким образом, чтобы контейнер оставался упорядоченным. Реализуйте функцию filter() из пункта 1.

Код 1.3. Абстрактный класс для связного списка LinkedListParent (функции push() и pop() чисто виртуальные) и IteratedLinkedList (введен механизм работы итераторов) и другие вспомогательные классы

```
#include <iostream>
#include <fstream>

using namespace std;

template <class T>
class Element
{
	//элемент связного списка
private:
	//указатель на предыдущий и следующий элемент
	Element* next;
	Element* prev;

	//информация, хранимая в поле
	T field;
public:
	Element(T value = 0, Element<T> * next_ptr = NULL, Ele-
ment<T> * prev_ptr = NULL)
	{
		field = value;
		next = next_ptr;
		prev - prev_ptr;
	}
	//доступ к полю *next
	virtual Element* getNext() { return next; }
	virtual void setNext(Element* value) { next = value; }

	//доступ к полю *prev
	virtual Element* getPrevious() { return prev; }
	virtual void setPrevious(Element* value) { prev = value; }

	//доступ к полю с хранимой информацией field
	virtual T getValue() { return field; }
	virtual void setValue(T value) { field = value; }

	template<class T> friend ostream& operator<< (ostream&
ustream, Element<T>& obj);
};

template<class T>
```

```
ostream& operator << (ostream& ustream, Element<T>& obj)
{
	ustream << obj.field;
	return ustream;
}

template <class T>
class LinkedListParent
{
protected:
	//достаточно хранить начало и конец
	Element<T>* head;
	Element<T>* tail;
	//для удобства храним количество элементов
	int num;
public:
	virtual int Number() { return num; }

	virtual Element<T>* getBegin() { return head; }

	virtual Element<T>* getEnd() { return tail; }

	LinkedListParent()
	{
		//конструктор без параметров
		cout << "\nParent constructor";
		head = NULL;
		num = 0;
	}

	//чисто виртуальная функция: пока не определимся с типом
списка, не сможем реализовать добавление
	virtual Element<T>* push(T value) = 0;

	//чисто виртуальная функция: пока не определимся с типом
списка, не сможем реализовать удаление
	virtual Element<T>* pop() = 0;

	virtual ~LinkedListParent()
	{
		//деструктор - освобождение памяти
		cout << "\nParent destructor";
	}
```

```
	//получение элемента по индексу - какова асимптотическая
оценка этого действия?
	virtual Element<T>* operator[](int i)
	{
		//индексация
		if (i<0 || i>num) return NULL;
		int k = 0;

		//ищем i-й элемент - вставем в начало и отсчитываем i
шагов вперед
		Element<T>* cur = head;
		for (k = 0; k < i; k++)
		{
			cur = cur->getNext();
		}
		return cur;
	}

	template<class T> friend ostream& operator<< (ostream&
ustream, LinkedListParent<T>& obj);
	template<class T> friend istream& operator>> (istream&
ustream, LinkedListParent<T>& obj);
};

template<class T>
ostream& operator << (ostream& ustream, LinkedListParent<T>&
obj)
{
	if (typeid(ustream).name() == typeid(ofstream).name())
	{
		ustream << obj.num << "\n";
		for (Element<T>* current = obj.getBegin(); current !=
NULL; current = current->getNext())
			ustream << current->getValue() << " ";
		return ustream;
	}

	ustream << "\nLength: " << obj.num << "\n";
	int i = 0;
	for (Element<T>* current = obj.getBegin(); current != NULL;
current = current->getNext(), i++)
		ustream << "arr[" << i << "] = " << current->getValue()
<< "\n";
```

```
	return ustream;
}

template<class T>
istream& operator >> (istream& ustream, LinkedListParent<T>&
obj)
{
	//чтение из файла и консоли совпадают
	int len;
	ustream >> len;
	//здесь надо очистить память под obj, установить obj.num = 0
	double v = 0;
	for (int i = 0; i < len; i++)
	{
		ustream >> v;
		obj.push(v);
	}
	return ustream;
}

template<typename ValueType>
class ListIterator : public
std::iterator<std::input_iterator_tag, ValueType>
{
private:

public:
	ListIterator() { ptr = NULL; }
	//ListIterator(ValueType* p) { ptr = p; }
	ListIterator(Element<ValueType>* p) { ptr = p; }
	ListIterator(const ListIterator& it) { ptr = it.ptr; }

	bool operator!=(ListIterator const& other) const { return
ptr != other.ptr; }
	bool operator==(ListIterator const& other) const { return
ptr == other.ptr; }//need for BOOST_FOREACH
	Element<ValueType>& operator*()
	{
		return *ptr;
	}
	ListIterator& operator++() { ptr = ptr->getNext(); return
*this; }
	ListIterator& operator++(int v) { ptr = ptr->getNext(); re-
turn *this; }
```

```
    ListIterator& operator=(const ListIterator& it) { ptr =
it.ptr; return *this; }
    ListIterator& operator=(Element<ValueType>* p) { ptr = p;
return *this; }
private:
    Element<ValueType>* ptr;
};

template <class T>
class IteratedLinkedList : public LinkedListParent<T>
{
public:
    IteratedLinkedList() : LinkedListParent<T>() { cout <<
"\nIteratedLinkedList constructor"; }
    virtual ~IteratedLinkedList() { cout <<
"\nIteratedLinkedList destructor"; }

    ListIterator<T> iterator;

    ListIterator<T> begin() { ListIterator<T> it =
LinkedListParent<T>::head; return it; }
    ListIterator<T> end() { ListIterator<T> it =
LinkedListParent<T>::tail; return it; }
};
```

Задание 1.4

Постройте итераторы для перемещения по списку. Переопределите функцию вывода содержимого списка с помощью итераторов. Итераторы двунаправленные.

Задание 1.5

Постройте шаблон класса списка D (из задания в пункте 3), который хранит объекты класса C (из задания в пункте 2), сохраняя упорядоченность по приоритету: полю или группе полей, указанных в варианте.