Модуль 4

Задача 8

Добавить механизм итераторов в класс **`TVector<>`**.

Продемонстрировать работу итератора при выводе значений элементов класса с помощью стандартного оператора цикла и цикла **range for**.

Задача 9

Добавить конструктор инициализации списком в класс **`TVector<>`**. Продемонстрировать работу.

Задача 10

Задан односвязный список. Общий функционал односвязного списка следующий.

```
template<typename LT>
class tlist
{
public:
	tlist(); // конструктор по умолчанию
	tlist( const tlist&tl ); // конструктор копирования
	tlist( tlist&&tl ); // конструктор перемещения
	~tlist();
protected:
	struct titem // элемент хранения
	{
		LT m_value; // значение данных
		titem*m_pNext; // ссылка на следующий элемент
		titem()
		{
			 m_value = LT( 0 ); // 0 для простых типов, пустой для объектов
			m_pNext = nullptr;
		};
	};
protected:
	titem*m_pFirst; // храним начало списка
protected:
	void init(); // функция инициализации
	void destroy( titem*pi ); // функция очистки элементов

public:
	int push_back( LT val ); // добавление в конец списка
	int size( void ); // вычисление размера
	int copy( const tlist&tl ); // копирование
	int move( tlist&tl ); // копирование перемещением
};

template<typename LT>
tlist<LT>::tlist()
{
	init();
}
template<typename LT>
tlist<LT>::tlist( const tlist<LT>&tl )
{
	init();
```

```
	copy( tl );
}
template<typename LT>
tlist<LT>::tlist( tlist<LT>&&tl )
{
	init();
	move( tl );
}

template<typename LT>
tlist<LT>::~tlist()
{
	destroy( m_pFirst );
}

template<typename LT>
void tlist<LT>::init()
{
	m_pFirst = nullptr;
}


template<typename LT>
void tlist<LT>::destroy( titem*pi )
{
	titem*p;
	while( pi ) // цикл по элементам (пока элемент существует) {
		p = pi->m_pNext; // запоминаем следующий элемент
		delete pi; // удаляем текущий элемент
		pi = p; // следующий стал текущим
	}
}


template<typename LT>
int tlist<LT>::push_back( LT val ) // добавление в конец списка
{
	titem*pLast, *pNew;

	pNew = new titem; // создаем элемент
	if( pNew ) // если элемент создался, то добавляем в конец {
		if( m_pFirst ) // проверяем, были ли элементы -> ищем последний {
			pLast = m_pFirst; // начинаем с первого
			    while( pLast->m_pNext ) // пока есть следующий
			{
				pLast = pLast->m_pNext; // следующий -> последний
			}
			 pLast->m_pNext = pNew; // нашли последний -> новый следующий
		}
		else // элементов не было -> новый первый {
			m_pFirst = pNew;
		}
		pNew->m_value = val; // сохраняем данные
		return size() - 1; // возвращаем номер добавленного элемента }
	return -1; // ошибка
}
template<typename LT>
int tlist<LT>::size( void ) // вычисление размера списка
{
	titem*p;
```

```
	int s = 0;
	p = m_pFirst; // начинаем с первого
	while( p ) // пока элементы есть
	{
		p = p->m_pNext; // следующий элемент
		s++; // добавляем счетчик
	}
	return s;
}

template<typename LT>
int tlist<LT>::copy( const tlist<LT>&lt ) // копирование
{
	if( this == &lt )return size(); // проверяем на a = a

	titem*p;
	tlist nl; // объект для безопасного копирования

	p = lt.m_pFirst; // начинаем с первого элемента
	while( p ) // пока элементы есть
	{
		if( nl.push_back( p->m_value ) < 0 )return 0;// добавляем значение
                                                     // если ошибка, то выходим
		p = p->m_pNext; // следующий элемент
	}
	return move( nl ); // готовые данные передаем в текущий объект
}
```

Реализовать функцию копирования перемещением `move()`. Добавить конструкторы инициализации массивом (см. ЛР2) и списком (см. ЛР9), операторы присваивания и функцию вывода элементов списка (см. ЛР2).

Продемонстрировать работу.

Задача 11.

Механизм итераторов реализован следующим образом.

```
template<typename LT>
class tlist
{
...
public:
	class iterator
	{
		friend class tlist; // основной класс имеет полный доступ к итератору
	protected:
		titem*m_pItem; // ссылка на элемент списка
	public:
		iterator() // конструктор по умолчанию – внешний доступ
		{
			m_pItem = nullptr; // по умолчанию ни на что не ссылается
		};
		iterator( const iterator&it ) // конструктор копирования - внешний доступ
		{
			m_pItem = it.m_pItem;
		}
	protected:
		iterator(titem*p) // конструктор инициализации – доступ только из списка {
```

```
                m_pItem = p;
            };
        public:
            LT& operator*() // оператор* - возвращает значение данных
            {
                return m_pItem->m_value;
            }
            iterator&operator++() // префиксный ++
            {
                m_pItem = m_pItem->m_pNext;
                return *this;
            }
            iterator operator++(int) // постфиксный ++
            {
                iterator it( *this );
                m_pItem = m_pItem->m_pNext;
                return it;
            }
            iterator&operator=( const iterator&it ) // присваивание копированием {
                m_pItem = it.m_pItem;
                return *this;
            }
            bool operator==(const iterator&it) // сравнение на равенство
            {
                return (m_pItem==it.m_pItem?true:false);
            }
            bool operator!=( const iterator&it ) // сравнение на неравенство
            {
                return (m_pItem!=it.m_pItem?true:false);
            }
        };
public:
        iterator begin();
        iterator end();
};
```

Реализовать функции **`begin()`** и **`end()`**.

Считая, что в списке хранится вектор, добавить функции скалярного произведения (внешняя функция) и нормы.

Продемонстрировать работу.

Задача 12.

Добавить функции:

- сумма и разность;

- прибавление и вычитание.

Продемонстрировать работу.