Отчет по работе Класс DoubleLinkedList

Выполнил: Смирнов Е. А., гр. 3530904/00006

- Методы для корректного копирования и перемещения:

1. Конструктор копирования

```
// Конструктор "копирования" – создание копии имеющегося списка
DoubleLinkedList::DoubleLinkedList(const DoubleLinkedList& src) {

	std::cout << "Вызвался копирование " << std::endl;
	head_ = nullptr;
	tail_ = nullptr;
	count_ = 0;

	// Голова списка, из которого копируем:
	Node* temp = src.head_;

	// Пока не конец списка:
	while (temp != nullptr)
	{   // Передираем данные:
		this->insertTail(temp->item_);
		temp = temp->next_;
	}
 }
```

2. Конструктор перемещения

```
// Конструктор "перемещения" принимаем в качестве параметра ссылку rvalue reference
DoubleLinkedList::DoubleLinkedList(DoubleLinkedList&& src) noexcept :
    count_(src.count_)
,   head_(src.head_)
,   tail_(src.tail_)
{
	std::cout << "Вызвался перемещение" << std::endl;

	src.count_ = 0;
	src.head_ = nullptr;
	src.tail_ = nullptr;
}
```

Для оператора копирования написана функция swap:

```
void DoubleLinkedList::swap(DoubleLinkedList& list){
	std::swap(head_, list.head_);
	std::swap(tail_, list.tail_);
	std::swap(count_, list.count_);
}
```

3. Оператор копирующего присваивания

```
DoubleLinkedList& DoubleLinkedList::operator=(DoubleLinkedList& list)
{
	std::cout << "Вызвался копирование =" << std::endl;
	DoubleLinkedList tmp(list);
	this->swap(tmp);
	return *this;
	}
```

4. Оператор перемещающего присваивания

```
DoubleLinkedList& DoubleLinkedList::operator=(DoubleLinkedList&& list) noexcept
	{
	std::cout << "Вызвался пермещение = " << std::endl;
	this->swap(list);
	list.head_ = nullptr;
	list.tail_ = nullptr;
	list.count_ = 0;
	return *this;
	}
```

- Методы, объявленные, но не реализованные в коде:

5.
```
// Вставить сформированный узел в хвост списка
void DoubleLinkedList::insertTail(Node* x)
{
	// task 5
	x->prev_ = tail_;
	if (tail_ == nullptr) {
		// список был пуст – новый элемент будет и первым, и последним
		head_ = x;
	}
	else {
		tail_->next_ = x;
	}
	tail_ = x;
	count_++;  // число элементов списка увеличилось
}
```

6.
```
// Замена информации узла на новое
DoubleLinkedList::Node* DoubleLinkedList::replaceNode(DoubleLinkedList::Node* x, int
item)
{
	// task 6
	x->item_ = item;
	return x;
}
```

7.
```
 // Удалить элемент из хвоста списка
bool DoubleLinkedList::deleteTail()
{
	  // task 3
	if (tail_ == nullptr) {
		return 0;  // список пуст, удалений нет
	}
	deleteNode(tail_);
	return 1;
}
```

8.
```
// Удаление узла с заданным значением
bool DoubleLinkedList::deleteItem(const int item)
{
	// task 8
	if (searchItem(item) == false) {
		return false;
	}
	else {
		Node* prevDdel = this->head_;
		while (prevDdel->item_ != item) {
			prevDdel = prevDdel->next_;
		}
		deleteNode(prevDdel);
	}
	return true;
}
```

9.
```
// Замена информации узла на новое
Возможно заполнить все значение новым, если передать all = true
bool DoubleLinkedList::replaceItem(int itemOld, int itemNew, bool all=false)
{
	// если нужно заменить на одно значение
	if (!all) {
		if (itemOld == itemNew) {
			return true;
		}
		else if (searchItem(itemOld) == false) {
			return false;
		}
		else {
			Node* prevDdel = this->head_;
			while (prevDdel->item_ != itemOld) {
				prevDdel = prevDdel->next_;
			}
			replaceNode(prevDdel, itemNew);
		}

	}
	else{
		Node* prevDdel = this->head_;
		while (prevDdel->next_!= nullptr) {
			replaceNode(prevDdel, itemNew);
			prevDdel = prevDdel->next_;
		}
		replaceNode(prevDdel, itemNew);
	}
	return true;
}
```

- Методы и функции, указанные в презентации «Практика #1»

Операция «» для вывода элементов от головы до хвоста
Дружественная функция

```
std::ostream& operator<<(std::ostream& out, DoubleLinkedList list)
{	DoubleLinkedList::Node* current = list.head_;	// Указатель на элемент
	while (current != nullptr) {
	out << current->item_ << ' ';
		current = current->next_;  // Переход к следующему элементу
	}
	out << std::endl;
	return out; }
```

Операция «==» для сравнения списков
Метод класса
Списки равны – если равны значения и порядок информационных частей

```
bool DoubleLinkedList::operator==(const DoubleLinkedList& list)
{
	Node* l = this->head_;
	Node* r = list.head_;
	if (this->count_ != list.count_) {
		return false;
	}
	else {
		if (l->item_ != r->item_) return false;
		while (l->item_ == r->item_ && l->next_!=nullptr) {
			if (l->item_ != r->item_) {
				return false;
			}
```

```
                l = l->next_; r = r->next_;
            }
        }
        return true;
}
```

Добавить в хвост исходного списка элементы списка, заданного параметром метода. Результат: модифицированный исходный список, *пустой список* (параметр метода). Список «второй» «*приклеивается*» к первому списку

```
void DoubleLinkedList::AddList(DoubleLinkedList& src) {
        if (src == *this) {

        }
        else {
                Node* noder = head_;
                while (noder->next_ != nullptr) {
                        noder = noder->next_;
                }

                Node* help = src.head_;
                noder->next_= help;

                tail_ = src.tail_;
                src.head_ = nullptr;
                src.tail_ = nullptr;
        }
 }
```

- Функция для тестирования:

```
void TestFunction()
{
        DoubleLinkedList list1;
        for (int i = 0; i < 11; i++)
        {list1.insertTail(i);}
        std::cout << "Вывод 1 список" << std::endl;
        std::cout << list1 << std::endl;

        DoubleLinkedList list;
        std::cout << "Вывод пустого списка" << std::endl;
        std::cout << list << std::endl;

        // присвоение ( копирование )
        DoubleLinkedList list2 = list1;
        std::cout << "Вывод 2 списока" << std::endl;
        std::cout << list2 << std::endl;

        // инициализация с r-value перемещение
        DoubleLinkedList list3 = DeleteHeadTail(list1);
        list1.outAll();
        list2.outAll();
        list3.outAll();

        // Оператор перемещающего присваивания
        list2 = DeleteHeadTail(list3);

        std::cout << "Одинаковые ли листы 1 и 2" << std::endl;
        std::cout << ((list1==list2) ? "--да" : "--нет") << std::endl;

        DoubleLinkedList list4;
        for (int i = 10; i >-1; i--)
        {list4.insertTail(i);}
```

```
        std::cout << "Замена одного элемента на другое" << std::endl;
        list4.replaceItem(4, 7);
        list4.outAll();

        std::cout << "Замена вссех элементов на новое" << std::endl;
        list4.replaceItem(4, 7, 1);
        list4.outAll();

        std::cout << "Склеивание списка" << std::endl;
        list4.AddList(list3);
        list4.outAll();
        list3.outAll();
}
```

Выходные значение после тестовой функции:

```
Вывод 1 список
0 1 2 3 4 5 6 7 8 9 10

Вывод пустого списка


Вывод 2 списока
0 1 2 3 4 5 6 7 8 9 10

0 1 2 3 4 5 6 7 8 9 10
0 1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9
Одинаковые ли листы 1 и 2
--нет
Замена одного элемента на другое
10 9 8 7 6 5 7 3 2 1 0
Замена вссех элементов на новое
7 7 7 7 7 7 7 7 7 7 7
Склеивание списка
7 7 7 7 7 7 7 7 7 7 7 1 2 3 4 5 6 7 8 9
```