LinkedList& LinkedList::operator=(LinkedList&& rhs) //перемещающее присваивание

{

// TODO+

clear();

swap(\*this, rhs);

return \*this;

}

bool LinkedList::operator ==(const LinkedList& other) const

{

// TODO+

if (this->size() == other.size())

{

Node\* left = other.first;

Node\* right = this->first;

while (left)

{

if (left->value != right->value)

{

return false;

}

left = left->next;

right = right->next;

}

return true;

}

else throw std::logic\_error("Error: not equal size of object !");

}

bool LinkedList::operator!=(const LinkedList& other) const

{

return !(\*this == other);

}

void LinkedList::erase(const Node\* node)//удаление узла из списка (8слайд)

{

// TODO+

if (node->previous)

node->previous->next = node->next;

else

first = node->next;

if (node->next)

node->next->previous = node->previous;

else

last = node->previous;

delete node;

--size\_;

}

void LinkedList::pop\_back()

{

// TODO+

erase(last);

}

void LinkedList::pop\_front()

{

// TODO+

erase(first);

}

LinkedList::Node\* LinkedList::insert\_after(LinkedList::Node\* after)

{

return nullptr;

}

LinkedList::Node\* LinkedList::insert\_before(LinkedList::Node\* before)

{

return nullptr;

}

void LinkedList::push\_back(const Data& value) //вставка элемента в конец списка

{

// TODO+

Node\* node = new Node;

node->value = value;

node->next = nullptr;

node->previous = last;

if (last != nullptr)

{

last->next = node;

}

last = node;

if (first == nullptr)

{

first = last = node;

}

++size\_;

}

void LinkedList::push\_front(const Data& value) //вставка элемента в начало списка

{

// TODO+

Node\* node = new Node; //создаем новый элемент

node->value = value; //вводим в него значение

node->previous = nullptr;

node->next = first;

if (first != nullptr)

{

first->previous = node;

}

first = node;

if (last == nullptr)

{

last = node;

}

++size\_;

}

void LinkedList::clear() //очистка списка

{

// TODO+

while (size\_)

erase (first);

}

void swap(LinkedList& left, LinkedList& right)

{

// TODO+

swap(left.first, right.first);

swap(left.last, right.last);

swap(left.size\_, right.size\_);

// HINT: Функция std::swap() меняет значения простых типов

}

LinkedList::LinkedList(): first {nullptr},last {nullptr},size\_ {0}

{

}

LinkedList::LinkedList(const std::initializer\_list<Data> values) :

LinkedList()

{

// По std::initializer\_list<T> возможен диапазонный for.

for (const Data& value : values)

{

push\_back(value);

}

}

LinkedList::LinkedList(const LinkedList& source) //копирование

: LinkedList()

{

// TODO+

Node\*current = source.first;

while (current)

{

push\_back(current->value);

current=current->next;

}

// HINT:

// Здесь нельзя воспользоваться диапазонным for, потому что на данном

// этапе задания у списка не реализованы итераторы, begin() и end().

}

LinkedList::LinkedList(LinkedList&& source) // перемещение

: LinkedList()

{

// TODO+

swap(\*this, source);

}

LinkedList::~LinkedList()

{

// TODO+

clear(); //пустой список

}

size\_t LinkedList::size() const

{

// TODO+

return size\_;

}

Data& LinkedList::value\_at(size\_t index) //??????

{

//TODO+

LinkedList::Node\* current = first;

size\_t current\_index = 0;

if (index >= size\_) throw std::out\_of\_range("Invalid index!");

else

{

while (current)

{

if (current\_index == index)

{

return current->value;;

}

++current\_index;

current = current->next;

}

}

}

void LinkedList::remove\_at(size\_t index)

{

// TODO+

Node\* node = (\*this).node\_at(index);

if (index >= size\_) throw std::out\_of\_range("Invalid index!");

else erase(node); //удаление

}

void LinkedList::insert\_before(size\_t index, const Data& value)

{

// TODO+

Node\* before = (\*this).node\_at(index); // TODO+

if (index >= size\_) throw std::out\_of\_range("Invalid index!");

else

{

if (before->previous == nullptr)

push\_front(value);

else

{

Node\* node = new Node; //создаем новый элемент

node->value = value; //вводим в него значение

node->previous = before->previous;

node->next = before;;

node->previous->next = node;

before->previous = node;

++size\_;

}

}

}

void LinkedList::insert\_after(size\_t index, const Data& value)

{

// TODO+

Node\* after = (\*this).node\_at(index);

if (index >= size\_) throw std::out\_of\_range("Invalid index!");

else

{

if (after->previous == nullptr)

push\_back(value);

else

{

Node\* node = new Node; //создаем новый элемент

node->value = value; //вводим в него значение

node->previous = after;

node->next = after->next;;

node->next->previous = node;

after->next = node;

++size\_;

}

}

}

LinkedList::Node\* LinkedList::node\_at(size\_t index)//поиск узла (9 слйад)

{

// TODO+

Node\* current = first;

size\_t current\_index = 0;

if (index >= LinkedList::size\_) throw std::out\_of\_range("Invalid index!");

else

{

while (current)

{

if (current\_index == index)

{

return current;

}

++current\_index;

current = current->next;

}

}

}

std::ostream& operator<<(std::ostream& output, const LinkedList& xs)//обход и печать списка (7слайд)

{

// TODO+

LinkedList::Node\* current = xs.first;

if (xs.size() == 0) output<<"[]";

else

{

output<<'[';

while (current)

{

output<<current -> value<<", ";;

current = current->next;

}

output<<']';

}

return output;

}