

Ring template class

Documentation

Author : Wiktor Łazarski

Index number : 281875

Field of study : Computer Science

Faculty : Electronics and Information Technology, Warsaw University of Technology

Table of content

1. General information 3

- 1.1 *Ring* template overview 3
- 1.2 Template parameters 3
- 1.3 Member types 4
- 1.4 Member functions 5
- 1.5 Non-member functions 6

2. Inner classes 7

- 2.1 *RingInvalidArgument* exception class 7
- 2.2 *Iterator* and *const_iterator* class 7
 - 2.3.1 Member types 7
 - 2.3.2 Member functions 7

3. Member and non-member functions implementation details 9

- 3.1 Member functions 9
 - 3.1.1 Standard member functions 9
 - 3.1.2 Operators 11
 - 3.1.3 Iterators 12
 - 3.1.4 Capacity 13
 - 3.1.5 Modifiers 14
 - 3.1.6 Operations 15
- 3.2 Non-member functions 17

4. Produce method 18

- 4.1 Implementation 18

5. Testing approach 19

- 5.1 *ErrorMessenger* class 19
- 5.2 Organisation of tests 20

1.1 *Ring* template overview

Ring class was implemented and compiled by MinGW for Windows compiler in CodeBlocks. Flags that were used during compilation are as follow :

```
mingw32-g++.exe -Wall -fexceptions -O2 -std=c++11
```

C++11 standard is highly recommended for proper use of *Ring* class template.

Link to full project : <https://github.com/Wiktos/Double-Linked-Ring>

Ring is a class template implemented as a circular, double linked list, abstract data structure. It supports constant complexity of insertion and removal of new elements, *Nodes*, from anywhere. Template class code is available in *ring.h* file. Implementation of all methods and *RingInvalidArgument* class are available in *ring.tpp*. *iterator* class implementation is available in *ring_iter.tpp*. To call *produce* method on two *Rings* you need to include *produce.h* file where this method is implemented.

1.2 Template parameters

```
template <typename Key, typename Info>
class Ring
```

Key	Typename of key by which <i>Node</i> will be recognized in <i>Ring</i> .
Info	Typename of data stored in particular <i>Node</i> .

1.3 Member types

Private member types :

Member type	Definition
<code>std::size_t</code> <code>length</code>	<i>length</i> private variable that contain information about current <i>Ring</i> size.

Protected member types :

Member type	Definition
<code>struct Node</code>	<i>Node</i> is a structure that contain Key value, Info value and pointer to next <i>Node</i> .
<code>Node *any</code>	<i>Node</i> pointer which prevents access to <i>Ring</i> .

Public member types :

Member type	Definition
<code>class RingInvalidArgument</code>	<i>RingInvalidArgument</i> is an exception class. <i>RingInvalidArgument</i> exceptions are thrown where member function is called with invalid arguments.
<code>class iterator</code>	Design pattern <i>iterator</i> class
<code>typedef const iterator const_iterator</code>	<i>const iterator</i> allows to perform only methods from <i>iterator</i> class which are <i>const</i> .

1.4 Member functions

Constructors :

<code>Ring() noexcept;</code>
<code>Ring(const Ring<Key, Info>& source);</code>
<code>Ring(Ring<Key, Info>&& source) noexcept;</code>
<code>Ring(std::initializer_list<std::pair<Key, Info>> ls);</code>

(All constructor are defined and implemented in header file)

Operators :

<code>Ring<Key, Info>& operator=(const Ring<Key, Info>& rhs);</code>
<code>Ring<Key, Info>& operator=(Ring<Key, Info>&& rhs) noexcept;</code>
<code>Ring<Key, Info> operator+(const Ring<Key, Info>& rhs) const;</code>
<code>Ring<Key, Info>& operator+=(const Ring<Key, Info>& rhs);</code>
<code>bool operator==(const Ring<Key, Info>& rhs) const noexcept;</code>
<code>bool operator!=(const Ring<Key, Info>& rhs) const noexcept;</code>

Iterators :

<code>iterator iter() noexcept;</code>
<code>const_iterator iter() const noexcept;</code>

Capacity :

<code>bool is_empty() const noexcept;</code>
<code>std::size_t size() const noexcept;</code>

Modifiers :

<code>void push(const Key& key, const Info& info;</code>
<code>void remove(const Key& loc);</code>
<code>void clear() noexcept;</code>
<code>void swap(Ring<Key, Info>& ring) ;</code>
<code>void reverse() noexcept;</code>

1.4 Member functions

Operations :

```
Ring<Key, Info> merge(const Ring<Key, Info>& ring) const;

bool compare(const Ring<Key, Info>& rhs, std::function<bool(const Ring<Key, Info>&,
const Ring<Key, Info>&)> comparator) const;

bool contain(const Key& loc) const;
```

Destructor :

```
~Ring() noexcept;
```

1.5 Non-member functions

Operator :

```
template <typename K, typename I>
friend std::ostream& operator<<(std::ostream& os, const Ring<K, I>& ring)

(private member function)
```

2.1 *RingInvalidArgument* exception class

RingInvalidArgument is an exception class that derive from *std::invalid_argument*. This class does not really extends properties of *std::invalid_argument* but just change its name. The purpose of having that class is to throw exception with different *typeid* than the *std::invalid_argument* one so it will be easier to define that method from *Ring* template class throws it.

Implementation :

```
template <typename Key, typename Info>
class Ring<Key, Info>::RingInvalidArgument final : public std::invalid_argument
{
public:
    using std::invalid_argument::invalid_argument;
};
```

2.2 *iterator* and *const_iterator* class

Iterator design pattern for *Ring*. Provide basic operators on *Node* pointer, which is hide in *private* section of that class.

2.2.1 Member types

Private member types :

Member type	Definition
<code>mutable Node *it</code>	<i>Node</i> pointer managed by <i>iterator</i> .

2.2.2 Member functions

Constructors :

<code>iterator(Node *node) noexcept;</code>
<code>iterator(const_iterator& source) noexcept;</code>
<code>iterator(iterator&& source) noexcept;</code>

(All constructor are defined and implemented in header file

Operators :

<code>iterator& operator=(const_iterator& rhs) noexcept;</code>
<code>iterator& operator=(iterator&& rhs) noexcept;</code>
<code>std::pair<Key&, Info&> operator*() noexcept;</code>
<code>std::pair<const Key&, const Info&> operator*() const noexcept;</code>
<code>std::pair<Key&, Info&>* operator->() noexcept;</code>
<code>const std::pair<Key&, Info&>* operator->() const noexcept;</code>
<code>bool operator==(iterator rhs) const noexcept;</code>
<code>bool operator!=(iterator rhs) const noexcept;</code>
<code>iterator operator++(int) const noexcept;</code>
<code>iterator& operator++() const noexcept;</code>
<code>iterator operator+(int rhs) const noexcept;</code>
<code>iterator operator--(int) const noexcept;</code>
<code>iterator& operator--() const noexcept;</code>
<code>iterator operator-(int rhs) const noexcept;</code>

Element access :

<code>Key& get_key() noexcept;</code>
<code>const Key& get_key() const noexcept;</code>
<code>Info& get_info() noexcept;</code>
<code>const Info& get_info() const noexcept;</code>

Destructor :

<code>~iterator() = default;</code>

To see the hole implementation of this class methods go to *ring_iter.hpp* file.

const_iterator is just a typedef name for *const iterator* :

```
typedef const iterator const_iterator;
```

Defining an object of *const_iterator* class allows you to call only those method of that class on that object which contain *const* at the end of declaration.

3.1 Member functions

3.1.1 Standard member functions

Default constructor
<code>Ring() noexcept;</code>
Parameters : none
Complexity : constant $O(1)$
Exception : exception safe
Notes : Assign <i>any</i> equals <i>nullptr</i> and <i>length</i> equals 0.

Copy constructor
<code>Ring(const Ring<Key, Info>& source);</code>
Parameters : source – constant reference to <i>Ring</i> copy pattern
Complexity : linear $O(n)$
Exception : <code>std::bad_alloc</code> may be thrown
Notes : Copy constructor call <i>operator=</i> .

Move constructor
<code>Ring(Ring<Key, Info>&& source) noexcept;</code>
Parameters : source – reference to <i>Ring</i> that will be moved to current created object
Complexity : constant $O(1)$
Exception : exception safe
Notes : Set source as an empty set.

3.1 Member functions

Constructor with <i>std::initializer_list</i>
<code>Ring(std::initializer_list<std::pair<Key, Info>> ls) noexcept;</code>
Parameters : ls – <i>std::initializer_list</i> object containing <i>std::pairs</i> of keys and infos that will be stored in <i>Ring</i>
Complexity : linear $O(n)$
Exception : exception safe
Notes : none

Destructor
<code>~Ring() noexcept;</code>
Parameters : none
Complexity : linear $O(n)$
Exception : exception safe
Notes : Call <i>clear</i> funtion that delete all allocated memory.

3.1 Member functions

3.1.2 Operators

Copy assignment operator =
<code>Ring<Key, Info>& operator=(const Ring<Key, Info>& rhs);</code>
Parameters : rhs – constant reference to <i>Ring</i> copy pattern
Complexity : linear $O(n)$
Exception : <code>std::bad_alloc</code> may be thrown
Notes : Return reference to <i>this</i> .

Move assignment operator =
<code>Ring<Key, Info>& operator=(Ring<Key, Info>&& rhs) noexcept;</code>
Parameters : rhs – reference to <i>Ring</i> that will be moved to our object
Complexity : constant $O(1)$
Exception : exception safe
Notes : Set rhs <i>Ring</i> as an empty set.

Add operator +
<code>Ring<Key, Info> operator+(const Ring<Key, Info>& rhs) const;</code>
Parameters : rhs – constant reference to <i>Ring</i> that will be merged with object
Complexity : linear $O(n)$
Exception : <code>std::bad_alloc</code> may be thrown
Notes : Method does not modify our object, call merge.

Add - equal operator +=
<code>Ring<Key, Info>& operator+=(const Ring<Key, Info>& rhs);</code>
Parameters : rhs – constant reference to <i>Ring</i> that will be added to the end of object
Complexity : linear $O(n)$
Exception : <code>std::bad_alloc</code> may be thrown
Notes : Call merge and assign operator =.

3.1 Member functions

Compare operator ==
<code>bool operator==(const Ring<Key, Info>& rhs) const noexcept;</code>
Parameters : rhs – constant reference to <i>Ring</i> that will compared with object
Complexity : linear $O(n)$
Exception : exception safe
Notes : First function compare sizes and then each <i>Node</i> .

Compare operator !=
<code>bool operator!=(const Ring<Key, Info>& rhs) const noexcept;</code>
Parameters : rhs – constant reference to <i>Ring</i> that will compared with object
Complexity : linear $O(n)$
Exception : exception safe
Notes : Returns negation of operator == comparison.

3.1.3 Iterators

Get <i>iterator</i> class object
<code>iterator iter() noexcept;</code>
Parameters : none
Complexity : constant $O(1)$
Exception : exception safe
Notes : Return <i>iterator</i> with pointer set at the beginning of <i>Ring</i> or return <i>iterator</i> with <i>Node*</i> set to <i>nullptr</i> if <i>Ring</i> is empty.

3.1 Member functions

Get <i>const_iterator</i> class object
<code>const_iterator iter() const noexcept;</code>
Parameters : none
Complexity : constant $O(1)$
Exception : exception safe
Notes : Return <i>const_iterator</i> with pointer set at the beginning of <i>Ring</i> or return <i>iterator</i> with <i>Node*</i> set to <i>nullptr</i> if <i>Ring</i> is empty.

3.1.4 Capacity

Check if <i>Ring</i> is empty
<code>bool is_empty() const noexcept;</code>
Parameters : none
Complexity : constant $O(1)$
Exception : exception safe
Notes : Check if <i>length</i> == 0.

Return <i>Ring's length</i>
<code>std::size_t size() const noexcept;</code>
Parameters : none
Complexity : constant $O(1)$
Exception : exception safe
Notes : Return current value of <i>length</i> .

3.1 Member functions

3.1.5 Modifiers

Push new <i>Node</i> to the <i>Ring</i>
<pre>void push(const Key& key, const Info& info);</pre>
Parameters : key – element's key that will be stored in new <i>Node</i> info – element's info that will be stored in new <i>Node</i>
Complexity : constant $O(1)$
Exception : <i>std::bad_alloc</i> may be thrown
Notes : none

Remove <i>Node</i> from <i>Ring</i> within the given Key
<pre>void remove(const Key& key);</pre>
Parameters : key – key that will be removed
Complexity : linear $O(n)$
Exception : <i>RingInvalidArgument</i> is thrown if <i>Ring</i> is empty.
Notes : none

Remove all <i>Nodes</i> from <i>Ring</i>
<pre>void clear() noexcept;</pre>
Parameters : none
Complexity : linear $O(n)$
Exception : exception safe
Notes : Clearing is permanent.

Swap <i>Rings</i>
<pre>void swap(Ring<Key, Info>& ring);</pre>
Parameters : ring – reference to <i>Ring</i> which will swap content with my object
Complexity : linear $O(n)$
Exception : <i>std::bad_alloc</i> may be thrown
Notes : Calls <i>operator=</i> .

3.1 Member functions

Reverse <i>prev</i> and <i>next</i> pointers of <i>Nodes</i>
<code>void reverse() noexcept;</code>
Parameters : none
Complexity : linear $O(n)$
Exception : exception safe
Notes : none

3.1.6 Operations

Merge <i>Rings</i>
<code>Ring<Key, Info> merge(const Ring<Key, Info> ring) const;</code>
Parameters : ring – <i>Ring</i> that will be added to the end of ours <i>Ring</i> object
Complexity : linear $O(n)$
Exception : <code>std::bad_alloc</code> may be thrown
Notes : Method does not modify our object.

Compare <i>Rings</i>
<code>bool compare(const Ring<Key, Info>& rhs, std::function<bool(const Ring<Key, Info>&, const Ring<Key, Info>&)> comparator) const;</code>
Parameters : rhs – right hand side of comparison comparator – specify the way <i>Rings</i> will be compared
Complexity : unknown because depends from comparator
Exception : unknown because depends from comparator
Notes : none

3.1 Member functions

Check if <i>Ring</i> contain <i>Node</i>
<code>bool contain(const Key& loc) const;</code>
Parameters : loc – key of element looked for
Complexity : linear $O(n)$
Exception : <i>RingInvalidArgument</i> is thrown if loc does not be found
Notes : none

3.2 Non-member functions – operator <<

Output stream operator <<
<pre>template <typename K, typename I> friend std::ostream& operator<<(std::ostream& os, const Ring<K, I>& ring);</pre>
Parameters : os – output stream ring – <i>Ring</i> object that will be inserted to os
Complexity : linear $O(n)$
Exception : unknown depends from os
Notes : none

4.1 Implementation

produce function is a iterative function. It is not a member function of *Ring* class. Method returns empty *Ring* if *num* parameter is invalid (less than 0). The result is create with the following algorithm .

<i>produce</i> function
<p>Full implementation :</p> <pre>template <typename Key, typename Info> Ring<Key, Info> produce(const Ring<Key, Info>& r1, int start1, int step1 , bool dir1, const Ring<Key, Info>& r2, int start2, int step2 , bool dir2, int num, bool dir) { Ring<Key, Info> retv; if(num < 0) return retv; typename Ring<Key, Info>::iterator iter1 = r1.iter(); typename Ring<Key, Info>::iterator iter2 = r2.iter(); //offset iter1 = iter1 + start1; iter2 = iter2 + start2; auto receive_values = [&retv](typename Ring<Key, Info>::const_iterator& iter, int start, int step , bool dir)->void{ for(int j = 0; j < step; j++){ std::pair<Key, Info> curr_val = *iter; retv.push(curr_val.first, curr_val.second); dir ? iter++ : iter--; } }; for(int i = 0; i < num; i++){ receive_values(iter1, start1, step1, dir1); receive_values(iter2, start2, step2, dir2); } if(!dir) retv.reverse(); return retv; }</pre>
<p>Parameters : r1 – first <i>Ring</i> start1 – starting point of first <i>Ring</i> step1 – number of elements taken from the first <i>Ring</i> dir1 – direction of iteration through first <i>Ring</i> r2 – second <i>Ring</i> start2 – starting point of second <i>Ring</i> step1 – number of elements taken from the second <i>Ring</i> dir1 – direction of iteration through first <i>Ring</i> num – number of algorithm calls dir – direction of iteration through returning <i>Ring</i></p>
<p>Complexity : linear $O(n)$</p>
<p>Exception : <i>std::bad_alloc</i> may be thrown</p>
<p>Notes : Function does not modify <i>Ring s1</i> or <i>Ring s2</i>.</p>

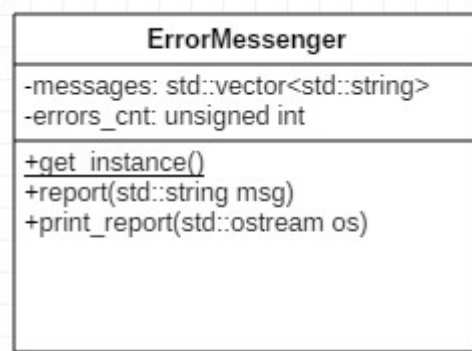
5.1 *ErrorMessenger* class

ErrorMessenger is a class implemented based on Singleton design pattern for collecting all fail tests information. Class belongs to *ring_test* namespace.

Contain two methods :

- **void** report(std::string msg) – function add messege to vector of messeges. Called when test fail.
- **void** print_report(std::ostream& os) – function called at the end of tests. To print number of errors and all messeges.

UML



Full implementation of *ErrorMessenger* class is placed in folder *./test/*.

5.2 Organisation of tests

All test are performed in *main.cpp* file. It could be find in *./source_code/* folder. Performing test code looks as follow :

```
using namespace std;

int main()
{
    ring_test::test_constructor();
    ring_test::test_push();
    ring_test::test_clear_and_assign_op();
    ring_test::test_contain_method();
    ring_test::test_compare_method();
    ring_test::test_equals_operator();
    ring_test::test_remove_method();
    ring_test::test_swap_method();
    ring_test::test_merge_and_binary_op();
    ring_test::test_iterator_class();
    ring_test::test_produce_method();

    ring_test::error_messenger().print_report(std::cout);
    return 0;
}
```

Implementation of particular test could be find in *./test/* folder.

All *Ring* test functions are declared in *ring_tests.h* and implementation of them is in file *ring_tests.cpp*. Tests belongs to namespace *ring_test*.

All *produce* function test method are declared in *ring_tests.h* and implementation of them is in file *ring_tests.cpp*. Tests belongs to namespace *ring_test*.