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

1.1 *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 suppors 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 *Node* will be recognized in *Ring*.
`Info`	Typename of data stored in particular *Node*.

1.3 Member types

Private member types :

Member type	Definition
`std::size_t length`	*length* private variable that contain information about current *Ring* size.

Protected member types :

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

Public member types :

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

1.4 Member functions

Constructors :

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

(All constructor are defined and implemented in header file)

Operators :

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

Iterators :

`iterator iter() noexcept;`
`const_iterator iter() const noexcept;`

Capacity :

`bool is_empty() const noexcept;`
`std::size_t size() const noexcept;`

Modifiers :

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

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
`mutable Node *it`	*Node* pointer managed by *iterator*.
`std::pair<Key&, Info&>* node_view`	Pair retured when *operator->* is called

2.2.2 Member functions

Constructors :

`iterator(Node *node) noexcept;`
`iterator(const_iterator& source) noexcept;`
`iterator(iterator&& source) noexcept;`

(All constructor are defined and implemented in header file

Operators :

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

Element access :

```
Key& get_key() noexcept;
const Key& get_key() const noexcept;
Info& get_info() noexcept;
const Info& get_info() const noexcept;
```

Destructor :

```
~iterator() = default;
```

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

const_iterator is just a typedefine 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
`Ring() noexcept;`
Parameters : none
Complexity : constant $O(1)$
Exception : exception safe
Notes : Assign *any* equals *nullptr* and *lenght* equals 0.

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

Move constructor
`Ring(Ring<Key, Info>&& source) noexcept;`
Parameters : source – reference to *Ring* 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 *std::initializer_list*
`Ring(std::initializer_list<std::pair<Key, Info>> ls) noexcept;`
Parameters : ls – *std::initializer_list* object containing *std::pair*s of keys and infos that will be stored in *Ring*
Complexity : linear $O(n)$
Exception : exception safe
Notes : none

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

3.1 Member functions

3.1.2 Operators

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

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

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

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

3.1 Member functions

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

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

3.1.3 Iterators

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

3.1 Member functions

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

3.1.4 Capacity

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

Return *Ring's length*
`std::size_t size() const noexcept;`
Parameters : none
Complexity : constant $O(1)$
Exception : exception safe
Notes : Return current value of *lenght.*

3.1 Member functions

3.1.5 Modifiers

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

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

Remove all *Nodes* from *Ring*
`void clear() noexcept;`
Parameters : none
Complexity : linear $O(n)$
Exception : exception safe
Notes : Clearing is pernament.

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

3.1 Member functions

Reverse *prev* and *next* pointers of *Nodes*
`void reverse() noexcept;`
Parameters : none
Complexity : linear $O(n)$
Exception : exception safe
Notes : none

3.1.6 Operations

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

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

3.1 Member functions

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

3.2 Non-member functions – operator <<

Output stream operator <<
`template <typename K, typename I>` `friend std::ostream& operator<<(std::ostream& os, const Ring<K, I>& ring);`
Parameters : os – output stream ring – *Ring* 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 .

***produce* function**

Full implementation :

```
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;
}
```

Parameters : r1 – first *Ring*
start1 – starting point of first *Ring*
step1 – number of elements taken from the first *Ring*
dir1 – direction of iteration through first *Ring*
r2 – second *Ring*
start2 – starting point of second *Ring*
step1 – number of elements taken from the second *Ring*
dir1 – direction of iteration through first *Ring*
num – number of algorithm calls
dir – direction of iteration through returning *Ring*

Complexity : linear O(n)

Exception : *std::bad_alloc* may be thrown

Notes : Function does not modify *Ring s1* or *Ring s2*.

5.1 *ErrorMessenger* class

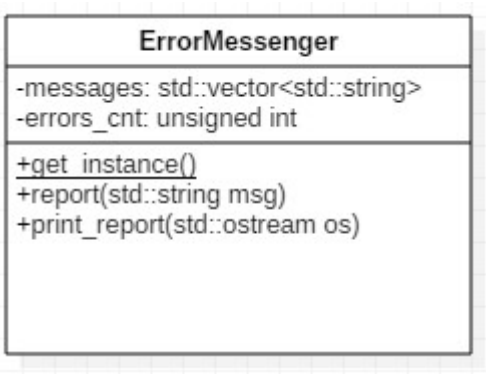
ErrorMessenger is a class implemented based on Singleton design pattern for collecting all fail tests information. Class belongs to *ring_test* namespace.

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*.