Ring Class Template
Documentation

0. **Author**

Name: Jan Radzimiński

Field of study: Computer Science, Faculty of Electronics and Information Technology

Index number: 293052

Group: 102

1. **General Information**

Doubly linked Ring project was implemented and compiled using Code Blocks and MinGW compiler, with C++11 standard.

Ring is a class template constructed as doubly linked ring. Ring consists of ‘’Nodes’’ which are elements of structure declared within the class. All struct elements consist of template type Key which works as ‘’ID’’ of the element, and Info which stores data in given ring element. Class supports all basic operation on the ring, such as insertion and deleting of elements with given key, printing ring, operators, data access etc. (all functions are described later in document). Class also has iterator implemented as class within class Ring. Iterator was especially useful while implementing non-class function Produce – function witch creates new Ring object from existing two other ring objects with some assumptions. Whole class implementation was placed in file “Ring.h’’ and Produce function in “Produce.h”. Tests were placed in “main.cpp” file.

2. **Ring Class Members**

Private:

Member:	**Explanation:**

struct Node:
Key ID;
Info Data;
Node *next;
Node *prev;
	Nodes are simply elements of the ring consisting of template type ‘Key ID’ which works like ID of the node, ‘Info Data’ to store data in the element, and pointer ‘next’ and ‘prev’ to next and previous element of the ring.

Node *any;	Pointer any is simply the pointer to any element of the ring. If ring is empty, it’s just a nullpointer.

int ring_size;	Integer consisting of number of elements in the ring, controlled by all add/remove methods. If ring is empty it is equal to 0.

Public:

Member:	**Explanation:**
class Iterator;	Iterator class, friend to the Ring class, which is used to navigate through ring by functions outside of the class. It allows to 'hide' all Node pointer values of the class so user using it does not have access to memory part of the class. More about this class can be found later in the document.

3. Ring Class Functions

Public:

Function:	**Explanation:**
Constructors:	
Ring<Key, Info>( ):	Empty constructor of the class, which assigns any pointer to null and ring size to 0.
~Ring<Key, Info>( );	Destructor of the class, which consist of Clear_Ring() method, which deletes all the nodes of the class and assures that there are no memory leaks after destruction of it.
Ring<Key, Info>(const Ring<Key, Info>& source);	Copy constructor of the class, which firstly set its pointer to null (for Clear() to work properly) and then uses assignment operator to copy source ring to itself.
Operators:	
Ring<Key, Info>& operator=(const Ring<Key, Info>& source);	Assignment operator, which firstly clears (this) ring to assure its empty, and then goes through Nodes of source ring and adds copy of each Node to 'this', using Push method.
Ring<Key, Info> operator+(const Ring<Key, Info>& source);	Add operator, which adds all Nodes from source ring using Push method.
Ring<Key, Info>& operator+=(const Ring<Key, Info>& source)	*this = *this + source;

bool operator==(const Ring<Key, Info>& source) const;	Boolean operator which returns true only if two rings have exact length and all nodes in both rings are identical and in the same order.
bool operator!=(const Ring<Key, Info>& source) const	returns !(*this==source);
bool operator>(const Ring<Key, Info>& source) const	Operator which returns true if 'this' ring size is bigger than source's.
bool operator<(const Ring<Key, Info>& source) const	Opposite as previous operator.
friend std::ostream& operator<<(std::ostream& os, const Ring<K, I>& seq);	Operator << which allows to print whole ring in different streams.
Node Add Functions:	
void **Push**(const Key &ID, const Info &Data);	Push function which adds new nodes with given Key and Info to ring and sets their pointers properly.
bool **Add_After_Key** (const Key &ID, const Info &Data, const Key &after_key);	Function which adds new Node after one Node with key same as 'after_key'. Returns true after successful creation of Node.
bool **Add_Before_Key** (const Key &ID, const Info &Data, const Key &bef_key);	Same as previous one, but adds node before given key.
Node Remove Functions:	
bool **Remove_By_Key** (const Key &rem_key);	Function that removes first founded Node with same key as rem_key. Returns true after successful deletion.
int **Remove_All_By_Key** (const Key &rem_key);	Function that removes all Nodes with given key and returns number of deleted keys.
void **Clear**();	Function that goes through every Node in the ring and deletes it. At the end it sets any to null and ring size to 0. Used in the destructor of class.
Ring Info:	
bool **Is_Empty**() const	Returns true if ring_size is 0 and false otherwise.
int **Ring_Length**() const	Returns ring_size.

Data Access:	
Info &**Get_Info**(const Key &nkey);	Goes through a ring and returns data of first node with key same as nkey. If such node weren’t found it throws an exception.
Info **&Any_Info**();	Returns Data of Node that is pointed by any and throws exception if ring Is empty.
Key &**Any_Key**();	Returns Key of Node that is pointed by any and throws exception if ring Is empty.
Other:	
void **Print_By_Key**(const Key &k) const	Prints (cout) all nodes with same key as k or nothing if such node doesn’t exist.
void **Reverse_Ring**();	Changes next pointers of all nodes into prev pointers, and vice versa. Finally changes any into any->next to make it more readable.
Iterators	
Iterator **ibegin**() const	Returns Iterator pointing to first Node or null if ring is empty.
const Iterator **const_ibegin**() const	Returns const Iterator pointing to first Node or null if ring is empty.
Iterator **iend**() const	Returns iterator pointing to any->prev or null if any = null.
const Iterator **const_iend**() const	Returns const iterator pointing any->prev or null if any = null.

Private:

//Remark: these are the functions which take or show Node pointers as arguments, used by public methods of class but not visible to user of the class.

Function:	**Explanation:**
void **Print_Node**(Node *curr) const;	Same as first in this table.
void **Print_Node**(Node *curr, int num) const;	Used during function testing also prints number of given node.
void **Print_Ring**() const;	Prints (cout) whole ring in larger scale than operator <<, with pointers next and prev

4. Iterator Class Members & Functions

Members (private):

Member:	**Explanation:**
Node *current;	‘Hidden’ pointer to element of the ring.

Functions:

Function:	**Explanation:**
Iterator();	Empty constructor, sets current to null.
Iterator(Node *ptr):	**Private** constructor which sets current to ptr. (available for Ring class because its friend of iterator class).
~Iterator()	Destructor of class.
Iterator(const Iterator& other)	Copy constructor sets this->currant as same as other.current .
Iterator& **operator**=(const Iterator& other)	Assignment operator sets this->currant as same as other.current .
bool **operator**==(const Iterator& source) const	Returns true if this and source current pointers are the same.
Iterator **operator**++(int)	Navigator of iterator, moves to next element in the ring (if its possible – otherwise it stays on null). //postfix one
Iterator **operator**++()	Same as previous but handling prefix.
Iterator **operator--**(int)	Navigator of iterator, moves to previous element in the ring (if its possible – otherwise it stays on null). //postfix one
Iterator **operator--**()	Same as previous but handling prefix.
Iterator **operator**+(int rhs)	Does incrementation rhs times.
Iterator **operator-**(int rhs)	Does decrementation rhs times.
Key &Show_Key() const	Returns key of node that current points to or throw exception if current=null;
Info &Show_Data() const	Returns Info of node that current points to or throw exception if current=null;

5. Produce Function

Ring<Key, Info> **Produce** (const Ring<Key, Info> &r1, int start1, int len1, bool dir1, const Ring<Key, Info> &r2, int start2, int len2, bool dir2, int repeat)	No class member function, returns another ring class object which is empty if any of len arguments is invalid (smaller then 0), or if repeat is 0 or if both r1 and r2 are empty. Otherwise in both r1 and r2 it starts taking elements from node ‘’int start’’ away from current any pointer of the ring (if starts is negative it goes backwards). Then for ,,repeat’’ times it adds len1 elements from r1 and len2 elements from r2 to r3. If dir1 or dir2 is false it moves backwards through given ring, otherwise it goes forward. Then it returns r3.