19.12.18

LABOLATORIES 2

[EADS] Algorithms and Data Structures

THE RING

DOUBLE LINKED RING

WITH ITERATOR

WIKTOR BARA

TASK

Design the template class, double linked ring ,and apply to class internal supporting class Iterator. Test the class test with external testing function – produce – that creates and returns new ring including data from nodes of two other given rings.

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);

r1, r2 – two given rings

start1, start2 – respectively the node of begin for first and second ring

len1, len2 – respectively number of nodes from data is taken in one cycle for first and second ring

dir1, dir2 – respectively the direction of changing nodes to next or previous for first and second ring

repeat – number of cycles

CLASS TEMPLATE

```
template<typename Key, typename Info>
class Ring{
private:
struct Node{
Key key;
Info info;
Node* next;
Node* prev;
};
Node* any;
int nr;
```

```
public:
class Iterator;
typedef const Iterator Const_Iterator;                    //supporting class

Ring();
Ring(const Ring &x);
~Ring();                                                  //constructors (destructor)

Ring &operator = (const Ring &x);                         //assignment operator
bool operator == (const Ring &x)const;                    //comparison operator
bool operator != (const Ring &x)const;
Ring operator + (const Ring &x)const;                     //plus operator

friend ostream& operator << (ostream &os, const Ring &x);   //display operator

bool isEmpty()const;                                      //returns true when  Ring has no nodes
void clearing();                                          //deletes all nodes from the ring
int howManyKeyExist (Key key)const;                       //returns number of nodes with given key
int howManyInfoExist (Info info)const;                    //returns number of nodes with given info
bool iteratorExists (const Iterator &x)const;             //returns true if given Iterator belongs to Ring

bool addNode (Key key, Info info);                        //add new node before 'any' node
bool addNodeByIterator (const Iterator &x);               //add new node with data given by Iterator before 'any'
                                                          node
bool addNodeAfterKey (Key key, Info info, Key where, int repeat);     //add new node after n-th(repeat) node
                                                                      with given key(where)
bool addNodeBeforeKey (Key key, Info info, Key where, int repeat);    //add new node before n-th(repeat) node
                                                                      with given key(where)
bool addNodeAfterInfo (Key key, Info info, Info where, int repeat);   //add new node after n-th(repeat) node
                                                                      with given info(where)
bool addNodeBeforeInfo (Key key, Info info, Info where, int repeat);  //add new node before n-th(repeat) node
                                                                      with given info(where)
bool addVectorOfNodes (const vector<Key> &x, const vector<Info> &y);      //add an amount of ordered
                                                                          nodes before 'any' node
//any addition method returns true if added at least one node
```

```
bool deleteFront();                    //deletes 'any' node

bool deleteBack();                     //deletes node before 'any' node

bool deleteByKey (Key where, int repeat);   //deletes n-th(repeat) node with given key(where)

bool deleteByIterator (const Iterator &x);    //deletes n-th(repeat) node with given info(where)


//any deleting method returns true if any node was deleted


vector<Key> getVectorOfKey()const;          //returns ordered vector of key

vector<Key> getVectorOfKeyByInfo (Info info)const;          //returns ordered vector of key that node->info is
                                                             the same as given

vector<Key> getVectorOfKeyFrom (int where, int number)const;        //returns ordered amount(number) of
                                                                     keys beginning from key that node is n-th(where)

vector<Info> getVectorOfInfo ()cosnt;        //returns ordered vector of info

vector<Info> getVectorOfInfoByKey (Key key)const;           //returns ordered vector of info that node->key is
                                                             the same as given

vector<Info> getVectorOfInfoFrom (int where, int number)const;       //returns ordered amount(number) of
                                                                      info beginning from info that node is n-th(repeat)


Iterator anything()const;                   //returns Iterator that node is 'any'

Const_Iterator anyconst()const;             //returns Const_Iterator that node is 'any'


int getNr()const;                           //returns size of Ring

void setNr(int nr);                         //sets size of Ring

};
```

SUPPORTING CLASS

```
class Iterator{
private:
   friend class Ring;
   mutable Node* current;

Iterator (Node *x);
public:
struct data{
   Key &key;
   Info &info;
};

Iterator();
Iterator (const Iterator &x);
~Iterator();                                  //constructors (destructor)

Iterator& operator = (const Iterator &x);     //assignment operator

data operator * ();
const data operator * () const;               //returns value of data

bool operator == (const Iterator &x)const;          //comparison operator
bool operator != (const Iterator &x)const;
Iterator operator ++ () const;                      //prefix changing to the next
Iterator operator ++ (int unused) const;            //postfix changing to the next
Iterator operator - - () const;                     //prefix changing to the previous
Iterator operator - - (int unused) const;           //postfix changing to the previous
Iterator operator += (int n) const;                 //changing to the next n times
Iterator operator -= (int n) const;                 //changing to the previous n times
};
```

TESTING CONDUCTED

Common case:

product(a, 0, 2, true , b ,3 ,3 ,true ,2);

The start1 or start2 is negative:

product(a, -2, 2, true, b, -3, 3, true, 2); it change start to the previous n-th node

The start1 or start2 is bigger than number of nodes in related Ring:

Product(a /*[10]*/ , 100, 2, true, b/*[5]*/, -40, 3, true, 2) the begin point is circulating in the loop

The length of copied nodes is negative or 0:

product(a, 2, -2, true, b, -3, 0, true, 2); the length is change to 0 so there are no nodes added

The length of copied nodes is bigger than number of nodes in the Ring:

Product(a/*[10]*/, 5, 100, true, b/*[5]*/, 3, 25, true, 1) an amount of copied nodes is closed in the loop

The dir1 or dir2 is false:

product(a, 2, 2, false, b, -3, 3, true, 2); the direction of taken nodes is reverse

The repeat is nonpositive:

Product(a, 2, 2, true, b, 0, 3, false, -3); the created Ring is empty

One or both Rings are empty:

product(empty1, 2, 2, true, b, 0, 3, false, 10); the empty Ring is neglected (if both returned Ring is also empty)