EADS LAB1
BiRing/Shuffle
Done by Filip Korzeniewski

General scheme of project.

The task was to create Bidirectional Circular List with typical functions. The functions are removing elements, adding elements, finding, changing data, getting data and some that made it easier implement. Moreover, there should have been used iterator to implement functions in easier and more clear way.

The second part of task was to create functions shuffle. Shuffle is to return Ring Type. It contents from 2 Rings given as parameters in special way. Third argument says how many elements in row it takes from first ring. Fourth argument is responsible for the same but with second ring. The last – fifth argument – says how many times this should be repeated. New ring is returned.

The outline:

Ring:

```
template<typename Key, typename Info>
class BiRing{
private:
    struct Node{
        Key key;
        Info info;
        Node* prev;
        Node* next;
        Node(Key _key, Info _info): key(_key), info(_info), prev(nullptr), next(nullptr) {};
    };
    Node* head;
    mutable Node* shuffleNode;
    int size;

    void createNode(Node* node, Key _key, Info _info);
    Node* DoesExist(Key _key);
public:
    void printRight();
    void printLeft();

    bool find_element(Key _key);
    Info getInfo(Key _key);
    bool setInfo(Key _key, Info _info);

    void pushRight(Key _key, Info _info);
    void pushLeft(Key _key, Info _info);
    bool pushAfter(Key AfterThisKey, Key _key, Info _info);

    bool pop_specific(Key KeyToDelete);

    void destroy_Ring();

    BiRing<Key, Info>& operator=(const BiRing<Key, Info>& x);
    BiRing<Key, Info>& operator+(const BiRing<Key, Info>& x);
    BiRing<Key, Info>& operator+=(const BiRing<Key, Info>& x);

    BiRing(const BiRing<Key, Info>& x);
    BiRing(): head(nullptr), shuffleNode(nullptr), size(0) {};
    ~BiRing(){ destroy_Ring(); };

    /* methods used for shuffle */
    BiRing<Key, Info> SubRing(int nb) const;
```

Iterator:

```
    /* ITERATOR */
    struct iterator{
        Node* iter;
        iterator(Node* temp): iter(temp) {};

        iterator& operator++() {
            iter = iter->next;
            return *this;
        };


        iterator& operator--() {
            iter = iter->prev;
            return *this;
        };

        bool operator==(const iterator& rhs){
            return iter == rhs.iter;
        };

        bool operator!=(const iterator& rhs){
            return iter!=rhs.iter;
        };

        Node& operator*()const{
            return *iter;
        };

        iterator& operator+=(int x){
            for(int i=0; i<x ;i++){
                iter = iter->next;
            }
            return *this;
        };

        iterator& operator-=(int x){
            for(int i=0; i<x ;i++){
                iter = iter->prev;
            }
            return *this;
        };
    };
};
```

Tests:

```
void BiRing_push_test();
void BiRing_pop_test();
void BiRing_operators_and_constructors_test();
void ShuffleTest();
```

Shuffle:

```
bool check_arguments(int nbfirst, int nbsecond, int reps);

template <typename Key, typename Info>
BiRing<Key, Info> shuffle(const BiRing<Key, Info>& first, const BiRing<Key, Info>& second, int nbfirst, int nbsecond, int reps){

    BiRing<Key, Info> to_return = BiRing<Key, Info>();

    if(check_arguments(nbfirst, nbsecond, reps) == false){
        std::cout << "WRONG PARAMETERS" <<std::endl;
        return to_return;
    }

    for(int i = 0; i < reps; i++){
        to_return += first.SubRing(nbfirst);
        to_return += second.SubRing(nbsecond);
    }
    return to_return;
};
```

Brief description

Shuffle uses function of Ring – Subring that creates subrings of Ring.

Check_arguments checks if arguments are right. Shuffle adds Subrings and returns the whole final version.

A few words about tests

Tests covered many cases that class BiRing and functions shuffle could failed.

Furthermore, and what is the most important – they show working of whole project.

Everything works as it should.