EADS LAB1
BiRing/Shuffle

Done by Filip Korzeniewski

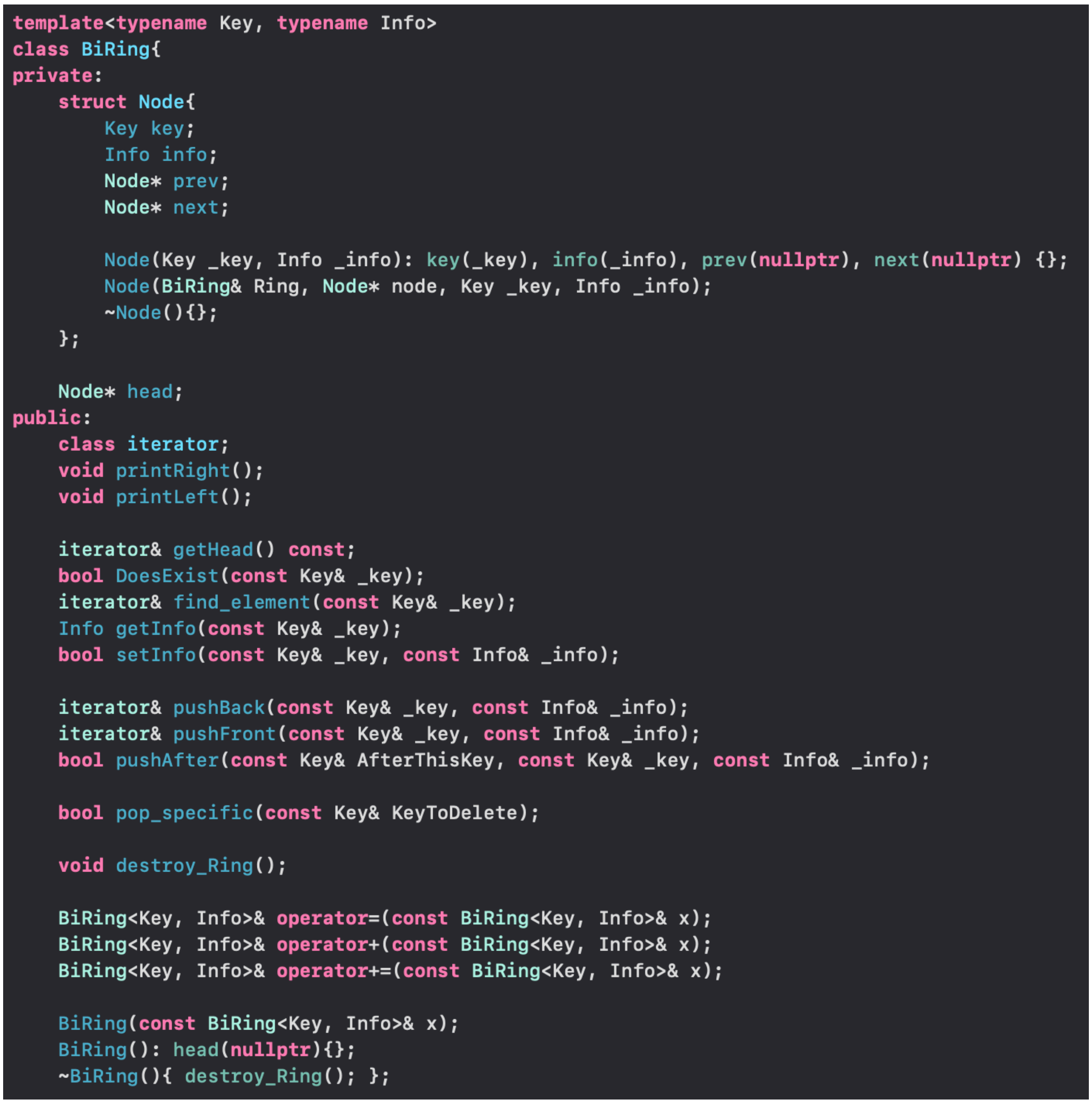
General scheme of project.

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.

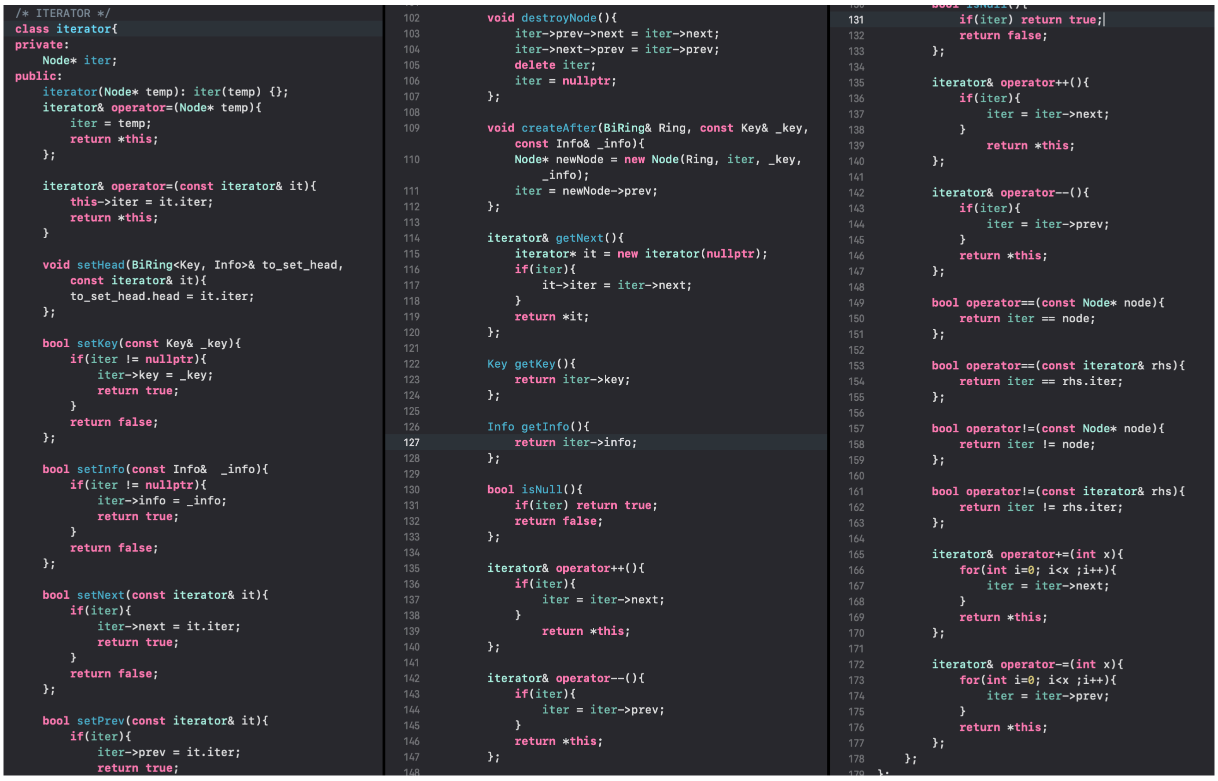
The outline:

Ring (without iterator):



```cpp
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(BiRing& Ring, Node* node, Key _key, Info _info);
        ~Node(){};
    };

    Node* head;
public:
    class iterator;
    void printRight();
    void printLeft();

    iterator& getHead() const;
    bool DoesExist(const Key& _key);
    iterator& find_element(const Key& _key);
    Info getInfo(const Key& _key);
    bool setInfo(const Key& _key, const Info& _info);

    iterator& pushBack(const Key& _key, const Info& _info);
    iterator& pushFront(const Key& _key, const Info& _info);
    bool pushAfter(const Key& AfterThisKey, const Key& _key, const Info& _info);

    bool pop_specific(const 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){};
    ~BiRing(){ destroy_Ring(); };
```

Iterator:



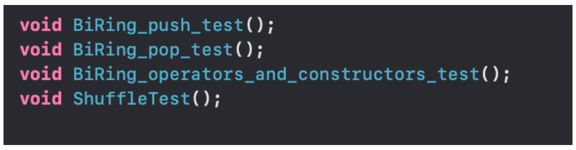
```cpp
/* ITERATOR */
class iterator{
private:
    Node* iter;
public:
    iterator(Node* temp): iter(temp) {};
    iterator& operator=(Node* temp){
        iter = temp;
        return *this;
    };

    iterator& operator=(const iterator& it){
        this->iter = it.iter;
        return *this;
    }

    void setHead(BiRing<Key, Info>& to_set_head,
        const iterator& it){
        to_set_head.head = it.iter;
    };

    bool setKey(const Key& _key){
        if(iter != nullptr){
            iter->key = _key;
            return true;
        }
        return false;
    };

    bool setInfo(const Info&  _info){
        if(iter != nullptr){
            iter->info = _info;
            return true;
        }
        return false;
    };

    bool setNext(const iterator& it){
        if(iter){
            iter->next = it.iter;
            return true;
        }
        return false;
    };

    bool setPrev(const iterator& it){
        if(iter){
            iter->prev = it.iter;
            return true;
```

```cpp
        void destroyNode(){
            iter->prev->next = iter->next;
            iter->next->prev = iter->prev;
            delete iter;
            iter = nullptr;
        };

        void createAfter(BiRing& Ring, const Key& _key,
            const Info& _info){
            Node* newNode = new Node(Ring, iter, _key,
                _info);
            iter = newNode->prev;
        };

        iterator& getNext(){
            iterator* it = new iterator(nullptr);
            if(iter){
                it->iter = iter->next;
            }
            return *it;
        };

        Key getKey(){
            return iter->key;
        };

        Info getInfo(){
            return iter->info;
        };

        bool isNull(){
            if(iter) return true;
            return false;
        };

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

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

```cpp
            if(iter) return true;
            return false;
        };

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

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

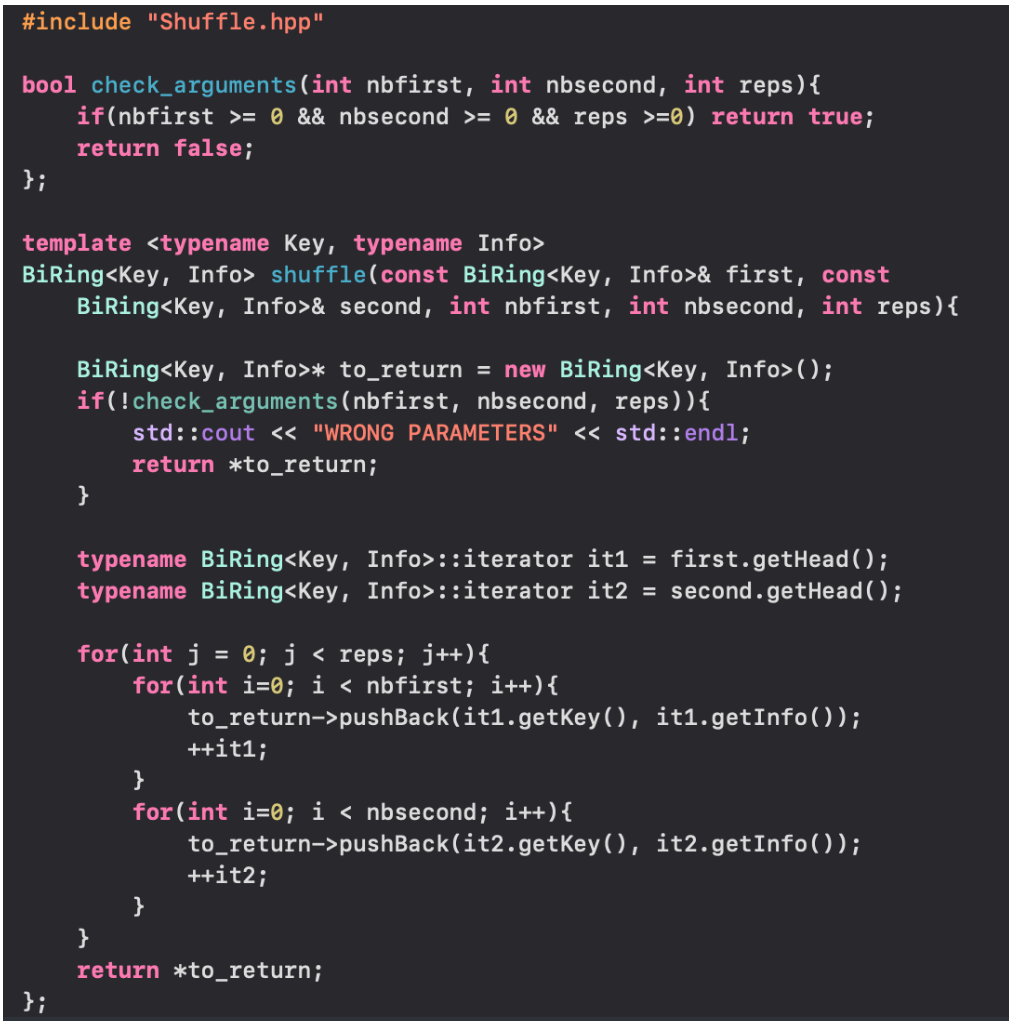
```cpp
        bool operator==(const Node* node){
            return iter == node;
        };

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

        bool operator!=(const Node* node){
            return iter != node;
        };

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



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

Shuffle:



```cpp
#include "Shuffle.hpp"

bool check_arguments(int nbfirst, int nbsecond, int reps){
    if(nbfirst >= 0 && nbsecond >= 0 && reps >=0) return true;
    return false;
};

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 = new BiRing<Key, Info>();
    if(!check_arguments(nbfirst, nbsecond, reps)){
        std::cout << "WRONG PARAMETERS" << std::endl;
        return *to_return;
    }

    typename BiRing<Key, Info>::iterator it1 = first.getHead();
    typename BiRing<Key, Info>::iterator it2 = second.getHead();

    for(int j = 0; j < reps; j++){
        for(int i=0; i < nbfirst; i++){
            to_return->pushBack(it1.getKey(), it1.getInfo());
            ++it1;
        }
        for(int i=0; i < nbsecond; i++){
            to_return->pushBack(it2.getKey(), it2.getInfo());
            ++it2;
        }
    }
    return *to_return;
};
```

Brief description

Shuffle checks if arguments are given correctly using additional function – check arguments which returns true if they are correct and false if not. Shuffle is based on iterators, they allow to “remember” last place of iteration.

A few words about tests

Tests covered many cases that class BiRing and functions shuffle could have failed. Furthermore, and what is the most important – they show working of whole project. Everything works as it should.

In whole project there is used iterator and many of its functions. Tests show that output is as it should be. It can be presumed that whole class iterator works correctly then.