Algorithms and Data Structures

EADS 2023Z LAB/102

Task #2 (term 2023Z)

Task #2 deadline for group LAB-102 is Wednesday, December 13th, 2023, 2 pm.

There are several steps to complete:

- Design bi_ring class according to a partial specification below.
- Use examples of *additional functions* to check if your class design is complete (i.e. you could implement all these additional functions using your bi_ring collection easily).
- You should implement your class template in the bi_ring.h file, and provide tests of your container in bi_ring_test.h and bi_ring_test.cpp files.
- During the lab class on Wednesday December 13th you'll receive a specification of an additional function to implement during this lab (this function will be similar to *additional functions*).
- You'll have 5 minutes to present your additional function implementation in the lab.
- You should upload your solution in the task defined in MS-Teams.
- I will review your bi_ring code after the lab class.

1 PART1 - DESIGN A CLASS

Design a class to represent collection (implemented as a doubly linked list). Write unit tests for the designed class, at least one test per method.

```
template <typename Key, typename Info>
class bi_ring      // implemented as doubly linked list
{
public:
      class iterator { /* … */ };
      class const_iterator { /* … */ };

      bi_ring();
      bi_ring(const bi_ring& src);
      ~bi_ring();
      bi_ring& operator=(const bi_ring& src);

      iterator push_front(const Key& key, const Info& info);
      iterator pop_front();

      iterator insert(iterator position, ...);
      iterator erase(iterator position);

      // what else can be useful in such a collection?
      // use examples of additional functions to guide you in the interface design
};
```

2 PART 2 – ADDITIONAL FUNCTIONS

Filtering (new object containing only the Key – Info pairs for which given predicate is true.

```
template<typename Key, typename Info>
bi_ring<Key, Info> filter(const bi_ring<Key, Info>& source,
                          bool (pred)(const Key&));

// source => [uno:1, due:2, tre:3, quattro:4, cinque:5, sei:6, sette:7, otto:8]
//
// filter<std::string, int>(source,
//                 [](const std::string& str) { return str.size() > 3; })
//   => [quattro:4, cinque:5, sette:7, otto:8]
```

Joining two rings with respect to Keys adding Info (Info must have operator + defined)

```
template<typename Key, typename Info>
bi_ring<Key, Info> join(const bi_ring<Key, Info>& first,
                          const bi_ring<Key, Info>& second);

// first => [uno:1, due:1, tre:2, quattro:1]
// second => [due:1, tre:1, quattro:3, cinque:5]
//
// join(first, second) => [uno:1, due:2, tre:3, quattro:4, cinque:5]
```

Elimination of repeated keys:

```
template<typename Key, typename Info>
bi_ring<Key, Info> unique(const bi_ring<Key, Info>& source,
                  Info(aggregate)(const Key&, const Info&, const Info&));

// source => [one: uno, two : due, three : tre, one : eins, two : zwei,
//                 three : drei, four : vier, five : cinque, six : sechs,
//                       seven : sieben, acht : otto ]
//
//unique<std::string, std::string>(src,
//    [](const std::string&, const std::string& i1, const std::string& i2)
//    {
//          return i1 + "-" + i2;
//    }
//);
//
// =>
// [ one : uno-eins, two : due-zwei, three : tre-drei,
//    four : vier, five : cinque, six : sechs, seven : sieben, acht : otto ]
```

Very strange join operation taking fcnt elements from the first ring and scnt elements from the second ring (repeated reps times)

```
template<typename Key, typename Info>
bi_ring<Key, Info> shuffle(
      const bi_ring<Key, Info>& first, unsigned int fcnt,
      const bi_ring<Key, Info>& second, unsigned int scnt,
      unsigned int reps);

// first => [uno:1, due:2, tre:3, quattro:4]
// second => [bir:1, iki:2, uc:3, dort:4, bes:5]
//
// shuffle(first, 1, second, 2, 3) =>
// [uno:1, bir:1, iki:2, due:2, uc:3, dort:4, tre:3, bes:5, bir:1]
```