franklist.h

```
#ifndef _FRANKLIST_HPP__
#define _FRANKLIST_HPP__

#include <iostream>

namespace vhuk {

template <typename T>
class FrankList;

template <typename T>
std::ostream& operator<<(std::ostream& out, const FrankList<T>& rhv);

template <typename T>
class FrankList
{
public:
    using value_type = T;
    using reference = value_type&;
    using const_reference = const value_type&;
    using size_type = std::size_t;
    using pointer = value_type*;
    using const_pointer = const value_type*;
private:
    struct Node
    {
        T val;
        Node* next;
        Node* prev;
        Node* asc;
        Node* desc;
        Node();
        Node(T val);
    };
private:
    class base_iterator
    {
        friend FrankList<value_type>;
    public:
        ~base_iterator();
        bool operator==(const base_iterator& rhv) const; //O(1)
        bool operator≠(const base_iterator& rhv) const; //O(1)
    protected:
        explicit base_iterator(Node* ptr); //O(1)
    protected:
        Node* ptr = nullptr;
    };
public:
    class const_iterator : public base_iterator
    {
        friend FrankList<value_type>;
    public:
        const_iterator(const base_iterator& rhv); //O(1)
        const_iterator(base_iterator&& rhv); //O(1)

        const const_iterator& operator=(const base_iterator& rhv); //O(1)
        const const_iterator& operator=(base_iterator&& rhv); //O(1)
```

```
58          const_reference operator*() const; //O(1)
59          const_pointer operator->() const; //O(1)
60
61          const const_iterator& operator++(); //O(1)
62          const const_iterator operator++(value_type); //O(1)
63          const const_iterator& operator--(); //O(1)
64          const const_iterator operator--(value_type); //O(1)
65
66      protected:
67          explicit const_iterator(Node* ptr); //O(1)
68      };
69
70  public:
71      class iterator : public const_iterator
72      {
73          friend FrankList<value_type>;
74      public:
75          iterator(const base_iterator& rhv); //O(1)
76          iterator(base_iterator&& rhv); //O(1)
77
78          reference operator*(); //O(1)
79          pointer operator->(); //O(1)
80
81          const iterator& operator=(const base_iterator& rhv); //O(1)
82          const iterator& operator=(base_iterator&& rhv); //O(1)
83
84
85      protected:
86          explicit iterator(Node* ptr); //O(1)
87      };
88
89  public:
90      class const_reverse_iterator : public base_iterator
91      {
92          friend FrankList<value_type>;
93      public:
94          const_reverse_iterator(const base_iterator& rhv); //O(1)
95          const_reverse_iterator(base_iterator&& rhv); //O(1)
96
97          const const_reverse_iterator& operator=(const base_iterator& rhv); //O(1)
98          const const_reverse_iterator& operator=(base_iterator&& rhv); //O(1)
99          const_reference operator*() const; //O(1)
100         const_pointer operator->() const; //O(1)
101
102         const const_reverse_iterator& operator++(); //O(1)
103         const const_reverse_iterator operator++(value_type); //O(1)
104         const const_reverse_iterator& operator--(); //O(1)
105         const const_reverse_iterator operator--(value_type); //O(1)
106
107     protected:
108         explicit const_reverse_iterator(Node* ptr); //O(1)
109     };
110 public:
111     class reverse_iterator : public const_reverse_iterator
112     {
113         friend FrankList<value_type>;
114     public:
115         reverse_iterator(const base_iterator& rhv); //O(1)
116         reverse_iterator(base_iterator&& rhv); //O(1)
117
```

```
118         reference operator*(); //O(1)
119         pointer operator->(); //O(1)
120 
121         const reverse_iterator& operator=(const base_iterator& rhv); //O(1)
122         const reverse_iterator& operator=(base_iterator&& rhv); //O(1)
123 
124 
125     protected:
126         explicit reverse_iterator(Node* ptr); //O(1)
127     };
128 public:
129     class const_asc_iterator : public base_iterator
130     {
131         friend FrankList<value_type>;
132     public:
133         const_asc_iterator(const base_iterator& rhv); //O(1)
134         const_asc_iterator(base_iterator&& rhv); //O(1)
135 
136         const const_asc_iterator& operator=(const base_iterator& rhv); //O(1)
137         const const_asc_iterator& operator=(base_iterator&& rhv); //O(1)
138         const_reference operator*() const; //O(1)
139         const_pointer operator->() const; //O(1)
140 
141         const const_asc_iterator& operator++(); //O(1)
142         const const_asc_iterator operator++(value_type); //O(1)
143         const const_asc_iterator& operator--(); //O(1)
144         const const_asc_iterator operator--(value_type); //O(1)
145 
146     protected:
147         explicit const_asc_iterator(Node* ptr); //O(1)
148     };
149 public:
150     class asc_iterator : public const_asc_iterator
151     {
152         friend FrankList<value_type>;
153     public:
154         asc_iterator(const base_iterator& rhv); //O(1)
155         asc_iterator(base_iterator&& rhv); //O(1)
156 
157         reference operator*(); //O(1)
158         pointer operator->(); //O(1)
159 
160         const asc_iterator& operator=(const base_iterator& rhv); //O(1)
161         const asc_iterator& operator=(base_iterator&& rhv); //O(1)
162 
163 
164     protected:
165         explicit asc_iterator(Node* ptr); //O(1)
166     };
167 public:
168     class const_desc_iterator : public base_iterator
169     {
170         friend FrankList<value_type>;
171     public:
172         const_desc_iterator(const base_iterator& rhv); //O(1)
173         const_desc_iterator(base_iterator&& rhv); //O(1)
174 
175         const const_desc_iterator& operator=(const base_iterator& rhv); //O(1)
176         const const_desc_iterator& operator=(base_iterator&& rhv); //O(1)
177         const_reference operator*() const; //O(1)
```

```
178         const_pointer operator→() const; //O(1)
179 
180         const const_desc_iterator& operator++(); //O(1)
181         const const_desc_iterator operator++(value_type); //O(1)
182         const const_desc_iterator& operator--(); //O(1)
183         const const_desc_iterator operator--(value_type); //O(1)
184 
185     protected:
186         explicit const_desc_iterator(Node* ptr); //O(1)
187     };
188 public:
189     class desc_iterator : public const_desc_iterator
190     {
191         friend FrankList<value_type>;
192     public:
193         desc_iterator(const base_iterator& rhv); //O(1)
194         desc_iterator(base_iterator&& rhv); //O(1)
195 
196         reference operator*(); //O(1)
197         pointer operator→(); //O(1)
198 
199         const desc_iterator& operator=(const base_iterator& rhv); //O(1)
200         const desc_iterator& operator=(base_iterator&& rhv); //O(1)
201 
202 
203     protected:
204         explicit desc_iterator(Node* ptr); //O(1)
205     };
206 public:
207     class const_multi_iterator : public base_iterator
208     {
209         friend FrankList<value_type>;
210     public:
211         const_multi_iterator(const base_iterator& rhv); //O(1)
212         const_multi_iterator(base_iterator&& rhv); //O(1)
213 
214         const const_multi_iterator& operator=(const base_iterator& rhv); //O(1)
215         const const_multi_iterator& operator=(base_iterator&& rhv); //O(1)
216         const_reference operator*() const; //O(1)
217         const_pointer operator→() const; //O(1)
218 
219         const const_multi_iterator& operator++(); //O(1)
220         const const_multi_iterator operator++(value_type); //O(1)
221         const const_multi_iterator& operator--(); //O(1)
222         const const_multi_iterator operator--(value_type); //O(1)
223 
224         void chmod(); //O(1)
225 
226     protected:
227         explicit const_multi_iterator(Node* ptr); //O(1)
228         bool mode = true;
229     };
230 public:
231     class multi_iterator : public const_multi_iterator
232     {
233         friend FrankList<value_type>;
234     public:
235         multi_iterator(const base_iterator& rhv); //O(1)
236         multi_iterator(base_iterator&& rhv); //O(1)
237 
```

```
238         reference operator*(); //O(1)
239         pointer operator->(); //O(1)
240 
241         const multi_iterator& operator=(const base_iterator& rhv); //O(1)
242         const multi_iterator& operator=(base_iterator&& rhv); //O(1)
243 
244 
245     protected:
246         explicit multi_iterator(Node* ptr); //O(1)
247     };
248 public:
249     class const_multi_reverse_iterator : public base_iterator
250     {
251         friend FrankList<value_type>;
252     public:
253         const_multi_reverse_iterator(const base_iterator& rhv); //O(1)
254         const_multi_reverse_iterator(base_iterator&& rhv); //O(1)
255 
256         const const_multi_reverse_iterator& operator=(const base_iterator& rhv);
    //O(1)
257         const const_multi_reverse_iterator& operator=(base_iterator&& rhv); //O(1)
258         const_reference operator*() const; //O(1)
259         const_pointer operator->() const; //O(1)
260 
261         const const_multi_reverse_iterator& operator++(); //O(1)
262         const const_multi_reverse_iterator operator++(value_type); //O(1)
263         const const_multi_reverse_iterator& operator--(); //O(1)
264         const const_multi_reverse_iterator operator--(value_type); //O(1)
265 
266 
267         void chmod(); //O(1)
268 
269     protected:
270         explicit const_multi_reverse_iterator(Node* ptr); //O(1)
271         bool mode = true;
272     };
273 public:
274     class multi_reverse_iterator : public const_multi_reverse_iterator
275     {
276         friend FrankList<value_type>;
277     public:
278         multi_reverse_iterator(const base_iterator& rhv); //O(1)
279         multi_reverse_iterator(base_iterator&& rhv); //O(1)
280 
281         reference operator*(); //O(1)
282         pointer operator->(); //O(1)
283 
284         const multi_reverse_iterator& operator=(const base_iterator& rhv); //O(1)
285         const multi_reverse_iterator& operator=(base_iterator&& rhv); //O(1)
286 
287 
288     protected:
289         explicit multi_reverse_iterator(Node* ptr); //O(1)
290     };
291 
292 public:
293     FrankList(); //O(1)
294     FrankList(size_type size); //O(n)
295     FrankList(size_type size, const_reference init); //O(n)
296     FrankList(const FrankList<value_type>& rhv); //O(n)
```

```
297     FrankList(FrankList<value_type>&& rhv); //O(1)
298     FrankList(std::initializer_list<value_type> init); //O(n)
299     template <typename input_iterator>
300     FrankList(input_iterator f, input_iterator l); //O(n)
301     ~FrankList();
302 
303 public:
304     void swap(FrankList<value_type>& rhv); //O(1)
305 
306     size_type size() const; //O(n)
307 
308     bool empty() const; //O(1)
309     void resize(size_type s, const_reference init = value_type()); //O(n)
310     void clear() noexcept; //O(n)
311 
312     void push_front(const_reference elem); //~O(1)
313     void pop_front(); //O(1)
314     void push_back(const_reference elem); //~O(1)
315     void pop_back(); //O(1)
316 
317     const_reference front() const; //O(1)
318     reference front(); //O(1)
319     const_reference back() const; //O(1)
320     reference back(); //O(1)
321     const_reference min() const; //O(1)
322     reference min(); //O(1)
323     const_reference max() const; //O(1)
324     reference max(); //O(1)
325 
326     const FrankList<value_type>& operator=(const FrankList<value_type>& rhv);
    //O(n)
327     const FrankList<value_type>& operator=(FrankList<value_type>&& rhv); //O(n)
328     const FrankList<value_type>& operator=(std::initializer_list<value_type> init)
    ; //O(n)
329 
330     bool operator==(const FrankList<value_type>& rhv) const; //O(n)
331     bool operator!=(const FrankList<value_type>& rhv) const; //O(n)
332     bool operator<(const FrankList<value_type>& rhv) const; //O(n)
333     bool operator<=(const FrankList<value_type>& rhv) const; //O(n)
334     bool operator>(const FrankList<value_type>& rhv) const; //O(n)
335     bool operator>=(const FrankList<value_type>& rhv) const; //O(n)
336 
337 public:
338     const_iterator cbegin() const; //O(1)
339     const_iterator cend() const; //O(1)
340     const_reverse_iterator crbegin() const; //O(1)
341     const_reverse_iterator crend() const; //O(1)
342     const_asc_iterator cabegin() const; //O(1)
343     const_asc_iterator caend() const; //O(1)
344     const_desc_iterator cdbegin() const; //O(1)
345     const_desc_iterator cdend() const; //O(1)
346     const_multi_iterator cmbegin() const; //O(1)
347     const_multi_iterator cmend() const; //O(1)
348     const_multi_iterator cmabegin() const; //O(1)
349     const_multi_iterator cmaend() const; //O(1)
350     const_multi_reverse_iterator cmrbegin() const; //O(1)
351     const_multi_reverse_iterator cmrend() const; //O(1)
352     const_multi_reverse_iterator cmrdbegin() const; //O(1)
353     const_multi_reverse_iterator cmrdend() const; //O(1)
354 
```

```
355     iterator begin(); //O(1)
356     iterator end(); //O(1)
357     reverse_iterator rbegin(); //O(1)
358     reverse_iterator rend(); //O(1)
359     asc_iterator abegin(); //O(1)
360     asc_iterator aend(); //O(1)
361     desc_iterator dbegin(); //O(1)
362     desc_iterator dend(); //O(1)
363     multi_iterator mbegin(); //O(1)
364     multi_iterator mend(); //O(1)
365     multi_iterator mabegin(); //O(1)
366     multi_iterator maend(); //O(1)
367     multi_reverse_iterator mrbegin(); //O(1)
368     multi_reverse_iterator mrend(); //O(1)
369     multi_reverse_iterator mrdbegin(); //O(1)
370     multi_reverse_iterator mrdend(); //O(1)
371 
372 public:
373     // template <typename iter>
374     // typename std::enable_if<std::is_base_of<const_iterator, iter>::value ||
375     //                         std::is_base_of<const_asc_iterator, iter>::value
    ||
376     //                         std::is_base_of<const_multi_iterator, iter>
    ::value,
377     //          iter>::type
378     // insert(iter pos, const_reference val) { //O(1)
379     //     return insert_def(pos, val);
380     // }
381 
382     // template <typename iter>
383     // typename std::enable_if<std::is_base_of<const_reverse_iterator, iter>
    ::value ||
384     //                         std::is_base_of<const_desc_iterator, iter>::value
    ||
385     //                         std::is_base_of<const_multi_reverse_iterator,
    iter>::value,
386     //          iter>::type
387     // insert(iter pos, const_reference val) { //O(1)
388     //     return insert_rev(pos, val);
389     // }
390 
391     template <typename iter>
392     typename std::enable_if<std::is_same<iterator, iter>::value ||
393                             std::is_same<asc_iterator, iter>::value ||
394                             std::is_same<multi_iterator, iter>::value,
395             iter>::type
396     insert(iter pos, const_reference val) { //O(1)
397         return insert_def(pos, val);
398     }
399 
400     template <typename iter>
401     typename std::enable_if<std::is_same<reverse_iterator, iter>::value ||
402                             std::is_same<desc_iterator, iter>::value ||
403                             std::is_same<multi_reverse_iterator, iter>::value,
404             iter>::type
405     insert(iter pos, const_reference val) { //O(1)
406         return insert_rev(pos, val);
407     }
408 
409     template <typename iter>
410     iter insert(iter pos, size_type size, const_reference val); //O(n)
```

```
411     template <typename iter>
412     iter insert(iter pos, std::initializer_list<value_type> init); //O(n)
413     template <typename iter, typename input_iterator>
414     iter insert(iter pos, input_iterator f, input_iterator l); //O(n)
415 
416     template <typename iter>
417     iter erase(iter pos); //O(1)
418     template <typename iter>
419     iter erase(iter f, iter l); //O(n)
420 
421     size_type remove(const_reference val); //O(n)
422     template <typename unary_predicate>
423     size_type remove_if(unary_predicate func); //O(n)
424 
425     void reverse(); //O(n)
426     void sort(bool reversed = false); //O(n)
427 
428     iterator find(const_reference elem); //O(n)
429     iterator rfind(const_reference elem); //O(n)
430 
431     template <typename unary_predicate, typename iter>
432     typename std::enable_if<std::is_same<iterator, iter>::value ||
433                             std::is_same<asc_iterator, iter>::value ||
434                             std::is_same<reverse_iterator, iter>::value ||
435                             std::is_same<desc_iterator, iter>::value,
436             void>::type
437     traverse(unary_predicate func, iter f, iter l)
438     {
439         for (auto i = f; i != l; i++)
440             func(*i);
441     }
442 
443     template <typename iter>
444     typename std::enable_if<std::is_same<iterator, iter>::value ||
445                             std::is_same<asc_iterator, iter>::value ||
446                             std::is_same<reverse_iterator, iter>::value ||
447                             std::is_same<desc_iterator, iter>::value,
448             void>::type
449     print(iter f, iter l)
450     {
451         traverse([](const_reference i){std::cout << i << "  ";}, f, l);
452     }
453 
454 
455     template <typename unary_predicate>
456     void traverse(unary_predicate func, bool sorted = false, bool reversed =
    false); //O(n)
457 
458     void print(bool sorted = false, bool reversed = false); //O(n)
459 
460 protected:
461     void put_in_sorted_order(Node* ptr); //O(n)
462     void organize_left(Node* ptr); //O(1)
463     void organize_right(Node* ptr); //O(1)
464 private:
465     template <typename iter>
466     iter insert_def(iter pos, const_reference val); //O(1)
467 
468     template <typename iter>
469     iter insert_rev(iter pos, const_reference val); //O(1)
```

```
470
471
472 private:
473     Node* head;
474     Node* tail;
475     Node* ahead;
476     Node* atail;
477 };
478
479
480
481 }
482
483 #include "franklist.hpp"
484
485 #endif // _FRANKLIST_HPP__
486
```