

Selected files

2 printable files

franklist.h
franklist.hpp

franklist.h

```
1  #ifndef FRANKLIST_H
2  #define FRANKLIST_H
3
4  #include <iostream>
5
6  namespace vhuk {
7
8  template <typename T>
9  class FrankList;
10
11  template <typename T>
12  std::ostream& operator<<(std::ostream& out, const FrankList<T>& rhv);
13
14  template <typename T>
15  class FrankList {
16  public:
17      using value_type = T;
18      using reference = value_type&;
19      using const_reference = const value_type&;
20      using size_type = std::size_t;
21      using pointer = value_type*;
22      using const_pointer = const value_type*;
23  private:
24      struct Node
25      {
26          T val;
27          Node* next;
28          Node* prev;
29          Node* asc;
30          Node* desc;
31          Node();
32          Node(T val);
33      };
34  private:
35      class base_iterator
36      {
37      friend FrankList<value_type>;
38      public:
39          ~base_iterator();
40          bool operator==(const base_iterator& rhv) const; //0(1)
41          bool operator!=(const base_iterator& rhv) const; //0(1)
42      protected:
43          explicit base_iterator(Node* ptr); //0(1)
44      protected:
45          Node* ptr = nullptr;
46      };
47  public:
48      class const_iterator : public base_iterator
```

```
49  {
50      friend FrankList<value_type>;
51  public:
52      const_iterator(const base_iterator& rhv); //0(1)
53      const_iterator(base_iterator&& rhv); //0(1)
54
55      const const_iterator& operator=(const base_iterator& rhv); //0(1)
56      const const_iterator& operator=(base_iterator&& rhv); //0(1)
57      const_reference operator*() const; //0(1)
58      const_pointer operator->() const; //0(1)
59
60      const const_iterator& operator++(); //0(1)
61      const const_iterator operator++(int); //0(1)
62      const const_iterator& operator--(); //0(1)
63      const const_iterator operator--(int); //0(1)
64
65  protected:
66      explicit const_iterator(Node* ptr); //0(1)
67  };
68
69  public:
70      class iterator : public const_iterator
71      {
72      friend FrankList<value_type>;
73      public:
74          iterator(const base_iterator& rhv); //0(1)
75          iterator(base_iterator&& rhv); //0(1)
76
77          reference operator*(); //0(1)
78          pointer operator->(); //0(1)
79
80          const iterator& operator=(const base_iterator& rhv); //0(1)
81          const iterator& operator=(base_iterator&& rhv); //0(1)
82      protected:
83          explicit iterator(Node* ptr); //0(1)
84      };
85
86  public:
87      class const_reverse_iterator : public base_iterator
88      {
89      friend FrankList<value_type>;
90      public:
91          const_reverse_iterator(const base_iterator& rhv); //0(1)
92          const_reverse_iterator(base_iterator&& rhv); //0(1)
93
94          const const_reverse_iterator& operator=(const base_iterator& rhv); //
95          const const_reverse_iterator& operator=(base_iterator&& rhv); //0(1)
96          const_reference operator*() const; //0(1)
97          const_pointer operator->() const; //0(1)
98
99          const const_reverse_iterator& operator++(); //0(1)
100         const const_reverse_iterator operator++(int); //0(1)
101         const const_reverse_iterator& operator--(); //0(1)
102         const const_reverse_iterator operator--(int); //0(1)
103
104     protected:
105         explicit const_reverse_iterator(Node* ptr); //0(1)
106     };
107 }
```

```

107 public:
108     class reverse_iterator : public const_reverse_iterator
109     {
110         friend FrankList<value_type>;
111     public:
112         reverse_iterator(const base_iterator& rhv); //0(1)
113         reverse_iterator(base_iterator&& rhv); //0(1)
114
115         reference operator*(); //0(1)
116         pointer operator->(); //0(1)
117
118         const reverse_iterator& operator=(const base_iterator& rhv); //0(1)
119         const reverse_iterator& operator=(base_iterator&& rhv); //0(1)
120
121     protected:
122         explicit reverse_iterator(Node* ptr); //0(1)
123     };
124 public:
125     class const_asc_iterator : public base_iterator
126     {
127         friend FrankList<value_type>;
128     public:
129         const_asc_iterator(const base_iterator& rhv); //0(1)
130         const_asc_iterator(base_iterator&& rhv); //0(1)
131
132         const const_asc_iterator& operator=(const base_iterator& rhv); //0(1)
133         const const_asc_iterator& operator=(base_iterator&& rhv); //0(1)
134         const_reference operator*() const; //0(1)
135         const_pointer operator->() const; //0(1)
136
137         const const_asc_iterator& operator++(); //0(1)
138         const const_asc_iterator operator++(int); //0(1)
139         const const_asc_iterator& operator--(); //0(1)
140         const const_asc_iterator operator--(int); //0(1)
141
142     protected:
143         explicit const_asc_iterator(Node* ptr); //0(1)
144     };
145 public:
146     class asc_iterator : public const_asc_iterator
147     {
148         friend FrankList<value_type>;
149     public:
150         asc_iterator(const base_iterator& rhv); //0(1)
151         asc_iterator(base_iterator&& rhv); //0(1)
152
153         reference operator*(); //0(1)
154         pointer operator->(); //0(1)
155
156         const asc_iterator& operator=(const base_iterator& rhv); //0(1)
157         const asc_iterator& operator=(base_iterator&& rhv); //0(1)
158
159     protected:
160         explicit asc_iterator(Node* ptr); //0(1)
161     };
162 public:
163     class const_desc_iterator : public base_iterator
164     {
165         friend FrankList<value_type>;

```

```

166 public:
167     const_desc_iterator(const base_iterator& rhv); //0(1)
168     const_desc_iterator(base_iterator&& rhv); //0(1)
169
170     const const_desc_iterator& operator=(const base_iterator& rhv); //0(1)
171     const const_desc_iterator& operator=(base_iterator&& rhv); //0(1)
172     const_reference operator*() const; //0(1)
173     const_pointer operator->() const; //0(1)
174
175     const const_desc_iterator& operator++(); //0(1)
176     const const_desc_iterator operator++(int); //0(1)
177     const const_desc_iterator& operator--(); //0(1)
178     const const_desc_iterator operator--(int); //0(1)
179
180     protected:
181         explicit const_desc_iterator(Node* ptr); //0(1)
182     };
183 public:
184     class desc_iterator : public const_desc_iterator
185     {
186         friend FrankList<value_type>;
187     public:
188         desc_iterator(const base_iterator& rhv); //0(1)
189         desc_iterator(base_iterator&& rhv); //0(1)
190
191         reference operator*(); //0(1)
192         pointer operator->(); //0(1)
193
194         const desc_iterator& operator=(const base_iterator& rhv); //0(1)
195         const desc_iterator& operator=(base_iterator&& rhv); //0(1)
196
197     protected:
198         explicit desc_iterator(Node* ptr); //0(1)
199     };
200 public:
201     class const_multi_iterator : public base_iterator
202     {
203         friend FrankList<value_type>;
204     public:
205         const_multi_iterator(const base_iterator& rhv); //0(1)
206         const_multi_iterator(base_iterator&& rhv); //0(1)
207
208         const const_multi_iterator& operator=(const base_iterator& rhv); //0(1)
209         const const_multi_iterator& operator=(base_iterator&& rhv); //0(1)
210         const_reference operator*() const; //0(1)
211         const_pointer operator->() const; //0(1)
212
213         const const_multi_iterator& operator++(); //0(1)
214         const const_multi_iterator operator++(int); //0(1)
215         const const_multi_iterator& operator--(); //0(1)
216         const const_multi_iterator operator--(int); //0(1)
217
218         void chmod(); //0(1)
219     protected:
220         explicit const_multi_iterator(Node* ptr); //0(1)
221         bool mode = true;
222     };
223 public:
224     class multi_iterator : public const_multi_iterator

```

```

225 {
226     friend FrankList<value_type>;
227 public:
228     multi_iterator(const base_iterator& rhv); //0(1)
229     multi_iterator(base_iterator&& rhv); //0(1)
230
231     reference operator*(); //0(1)
232     pointer operator->(); //0(1)
233
234     const multi_iterator& operator=(const base_iterator& rhv); //0(1)
235     const multi_iterator& operator=(base_iterator&& rhv); //0(1)
236
237 protected:
238     explicit multi_iterator(Node* ptr); //0(1)
239 };
240 public:
241     class const_multi_reverse_iterator : public base_iterator
242     {
243         friend FrankList<value_type>;
244     public:
245         const_multi_reverse_iterator(const base_iterator& rhv); //0(1)
246         const_multi_reverse_iterator(base_iterator&& rhv); //0(1)
247
248         const const_multi_reverse_iterator& operator=(const base_iterator& rhv)
249 ; //0(1)
250         const const_multi_reverse_iterator& operator=(base_iterator&& rhv); //
0(1)
251         const_reference operator*() const; //0(1)
252         const_pointer operator->() const; //0(1)
253
254         const const_multi_reverse_iterator& operator++; //0(1)
255         const const_multi_reverse_iterator operator++(int); //0(1)
256         const const_multi_reverse_iterator& operator--(); //0(1)
257         const const_multi_reverse_iterator operator--(int); //0(1)
258
259         void chmod(); //0(1)
260     protected:
261         explicit const_multi_reverse_iterator(Node* ptr); //0(1)
262         bool mode = true;
263     };
264 public:
265     class multi_reverse_iterator : public const_multi_reverse_iterator
266     {
267         friend FrankList<value_type>;
268     public:
269         multi_reverse_iterator(const base_iterator& rhv); //0(1)
270         multi_reverse_iterator(base_iterator&& rhv); //0(1)
271
272         reference operator*(); //0(1)
273         pointer operator->(); //0(1)
274
275         const multi_reverse_iterator& operator=(const base_iterator& rhv); //
0(1)
276         const multi_reverse_iterator& operator=(base_iterator&& rhv); //0(1)
277
278     protected:
279         explicit multi_reverse_iterator(Node* ptr); //0(1)
280     };

```

```

281 public:
282     FrankList(); //0(1)
283     FrankList(size_type size); //0(n)
284     FrankList(size_type size, const_reference init); //0(n)
285     FrankList(const FrankList<value_type>& rhv); //0(n)
286     FrankList(FrankList<value_type>&& rhv); //0(1)
287     FrankList(std::initializer_list<value_type> init); //0(n)
288     template <typename input_iterator>
289     FrankList(input_iterator f, input_iterator l); //0(n)
290     ~FrankList();
291
292 public:
293     void swap(FrankList<value_type>& rhv); //0(1)
294
295     size_type size() const; //0(n)
296
297     bool empty() const; //0(1)
298     void resize(size_type s, const_reference init = value_type()); //0(n)
299     void clear() noexcept; //0(n)
300
301     void push_front(const_reference elem); //~0(1)
302     void pop_front(); //0(1)
303     void push_back(const_reference elem); //~0(1)
304     void pop_back(); //0(1)
305
306     const_reference front() const; //0(1)
307     reference front(); //0(1)
308     const_reference back() const; //0(1)
309     reference back(); //0(1)
310     const_reference min() const; //0(1)
311     reference min(); //0(1)
312     const_reference max() const; //0(1)
313     reference max(); //0(1)
314
315     const FrankList<value_type>& operator=(const FrankList<value_type>& rhv);
316 //0(n)
317     const FrankList<value_type>& operator=(FrankList<value_type>&& rhv); //0(n)
318     const FrankList<value_type>& operator=(std::initializer_list<value_type>
init); //0(n)
319
320     bool operator==(const FrankList<value_type>& rhv) const; //0(n)
321     bool operator!=(const FrankList<value_type>& rhv) const; //0(n)
322     bool operator<(const FrankList<value_type>& rhv) const; //0(n)
323     bool operator<=(const FrankList<value_type>& rhv) const; //0(n)
324     bool operator>(const FrankList<value_type>& rhv) const; //0(n)
325     bool operator>=(const FrankList<value_type>& rhv) const; //0(n)
326
327 public:
328     const_iterator cbegin() const; //0(1)
329     const_iterator cend() const; //0(1)
330     const_reverse_iterator crbegin() const; //0(1)
331     const_reverse_iterator crend() const; //0(1)
332     const_asc_iterator cabegin() const; //0(1)
333     const_asc_iterator caend() const; //0(1)
334     const_desc_iterator cdbegin() const; //0(1)
335     const_desc_iterator cdend() const; //0(1)
336     const_multi_iterator cmbegin() const; //0(1)
337     const_multi_iterator cmend() const; //0(1)

```

```

338     const_multi_iterator cmbegin() const; //0(1)
339     const_multi_iterator cmaend() const; //0(1)
340     const_multi_reverse_iterator cmrbegin() const; //0(1)
341     const_multi_reverse_iterator cmrend() const; //0(1)
342     const_multi_reverse_iterator cmrdbegin() const; //0(1)
343     const_multi_reverse_iterator cmrdend() const; //0(1)
344
345     iterator begin(); //0(1)
346     iterator end(); //0(1)
347     reverse_iterator rbegin(); //0(1)
348     reverse_iterator rend(); //0(1)
349     asc_iterator abegin(); //0(1)
350     asc_iterator aend(); //0(1)
351     desc_iterator dbegin(); //0(1)
352     desc_iterator dend(); //0(1)
353     multi_iterator mbegin(); //0(1)
354     multi_iterator mend(); //0(1)
355     multi_iterator mabegin(); //0(1)
356     multi_iterator maend(); //0(1)
357     multi_reverse_iterator mrbegin(); //0(1)
358     multi_reverse_iterator mrend(); //0(1)
359     multi_reverse_iterator mrdbegin(); //0(1)
360     multi_reverse_iterator mrdend(); //0(1)
361
362 public:
363     template <typename iter>
364     typename std::enable_if<std::is_base_of<const_iterator, iter>::value ||
365     ||
366     std::is_base_of<const_multi_iterator, iter>
367     ::value,
368     iter>::type
369     insert(iter pos, const_reference val) { //0(1)
370         return insert_def(pos, val);
371     }
372
373     template <typename iter>
374     typename std::enable_if<std::is_base_of<const_reverse_iterator, iter>
375     ::value ||
376     ||
377     std::is_base_of<const_desc_iterator, iter>::value
378     ||
379     std::is_base_of<const_multi_reverse_iterator,
380     iter>::value,
381     iter>::type
382     insert(iter pos, const_reference val) { //0(1)
383         return insert_rev(pos, val);
384     }
385
386     template <typename iter>
387     iter insert(iter pos, size_type size, const_reference val); //0(n)
388
389     template <typename iter>
390     iter insert(iter pos, std::initializer_list<value_type> init); //0(n)
391
392     template <typename iter, typename input_iterator>
393     iter insert(iter pos, input_iterator f, input_iterator l); //0(n)
394
395     template <typename iter>
396     iter erase(iter pos); //0(1)
397
398     template <typename iter>
399     iter erase(iter f, iter l); //0(n)

```

```

393     size_type remove(const_reference val); //0(n)
394     template <typename unary_predicate>
395     size_type remove_if(unary_predicate func); //0(n)
396
397     void reverse(); //0(n)
398     void sort(bool reversed = false); //0(n)
399
400     iterator find(const_reference elem); //0(n)
401     iterator rfind(const_reference elem); //0(n)
402
403     template <typename unary_predicate>
404     void traverse(unary_predicate func, bool sorted = false, bool reversed =
405     false); //0(n)
406
407     void print(bool sorted = false, bool reversed = false); //0(n)
408
409 protected:
410     void put_in_sorted_order(Node* ptr); //0(n)
411     void organize_left(Node* ptr); //0(1)
412     void organize_right(Node* ptr); //0(1)
413
414 private:
415     template <typename iter>
416     iter insert_def(iter pos, const_reference val); //0(1)
417
418     template <typename iter>
419     iter insert_rev(iter pos, const_reference val); //0(1)
420
421 private:
422     Node* head;
423     Node* tail;
424     Node* ahead;
425     Node* atail;
426
427 };
428
429 #include "franklist.hpp"
430
431 #endif // _FRANKLIST_H_
432
433
434

```

franklist.hpp

```

1  #ifndef FRANKLIST_HPP
2  #define FRANKLIST_HPP
3  #include "franklist.h"
4
5
6  namespace vhuk {
7
8
9  template <typename T>
10 std::ostream& operator<<(std::ostream& out, const FrankList<T>& rhv) {
11     for (auto it = rhv.cbegin(); it != rhv.cend(); ++it) {
12         out << *it << " ";

```

```

13     }
14     return out;
15 }
16
17 template <typename T>
18 FrankList<T>::Node::Node() : val(), next(nullptr), prev(nullptr), asc(nullptr), d
19
20 template <typename T>
21 FrankList<T>::Node::Node(T val) : val(val), next(nullptr), prev(nullptr), asc(nu
22
23 template <typename T>
24 FrankList<T>::FrankList() : head(nullptr), tail(nullptr), ahead(nullptr), atail(
25
26 template <typename T>
27 FrankList<T>::FrankList(std::size_t size) : FrankList() {
28     for (std::size_t i = 0; i < size; ++i) {
29         push_back(T());
30     }
31 }
32
33 template <typename T>
34 FrankList<T>::FrankList(std::size_t size, const_reference init) : FrankList() {
35     for (std::size_t i = 0; i < size; ++i) {
36         push_back(init);
37     }
38 }
39
40 template <typename T>
41 FrankList<T>::FrankList(const FrankList<T>& rhv) : FrankList() {
42     for (auto it = rhv.begin(); it != rhv.end(); ++it) {
43         push_back(*it);
44     }
45 }
46
47 template <typename T>
48 FrankList<T>::FrankList(FrankList<T>&& rhv) : head(rhv.head), tail(rhv.tail), ah
49 {
50     rhv.head = nullptr;
51     rhv.tail = nullptr;
52     rhv.ahead = nullptr;
53     rhv.atail = nullptr;
54 }
55
56 template <typename T>
57 FrankList<T>::FrankList(std::initializer_list<value_type> init) : FrankList() {
58     for (const auto& i : init){
59         push_back(i);
60     }
61 }
62 template <typename input_iterator>
63 FrankList<T>::FrankList(input_iterator f, input_iterator l) : FrankList() {
64     for (input_iterator it = f; it != l; ++it){
65         push_back(*it);
66     }
67 }
68
69 template <typename T>
70 FrankList<T>::~~FrankList() {

```

```

71     clear();
72 }
73
74 template <typename T>
75 void FrankList<T>::swap(FrankList<T>& rhv) {
76     std::swap(head, rhv.head);
77     std::swap(tail, rhv.tail);
78     std::swap(ahead, rhv.ahead);
79     std::swap(atail, rhv.atail);
80 }
81
82 template <typename T>
83 typename FrankList<T>::size_type FrankList<T>::size() const {
84     size_type size = 0;
85     Node* i = head;
86     while (i != nullptr){
87         ++size;
88         i = i -> next;
89     }
90
91     return size;
92 }
93
94 template <typename T>
95 bool FrankList<T>::empty() const {
96     return head == nullptr;
97 }
98
99 template <typename T>
100 void FrankList<T>::resize(size_type s, const_reference init){
101     size_type size = this -> size();
102     if (size > s){
103         for (size_type i = s; i < size; ++i){
104             pop_back();
105         }
106     } else if (size < s) {
107         for (size_type i = size; i < s; ++i) {
108             push_back(init);
109         }
110     }
111 }
112
113 template <typename T>
114 void FrankList<T>::clear () noexcept {
115     Node* i = head;
116     while (i != nullptr) {
117         Node* next = i->next;
118         delete i;
119         i = next;
120     }
121
122     head = nullptr;
123     tail = nullptr;
124     ahead = nullptr;
125     atail = nullptr;
126 }
127
128 template <typename T>
129 void FrankList<T>::push_front(const_reference elem) {

```

```

130 Node* ptr = new Node(elem);
131 if (head != nullptr) {
132     head -> prev = ptr;
133 } else {
134     tail = ptr;
135 }
136
137 ptr -> next = head;
138 head = ptr;
139 put_in_sorted_order(ptr);
140 }
141
142 template <typename T>
143 void FrankList<T>::pop_front() {
144     if (head == nullptr) {
145         return;
146     }
147
148     if (head -> next != nullptr) {
149         head -> next -> prev = nullptr;
150         if (head -> asc && head -> desc){
151             head -> desc -> asc = head -> asc;
152             head -> asc -> desc = head -> desc;
153         }
154         Node* ptr = head -> next;
155         delete head;
156         head = ptr;
157     } else {
158         delete head;
159         head = nullptr;
160         tail = nullptr;
161     }
162 }
163
164 template <typename T>
165 void FrankList<T>::push_back(const_reference elem) {
166     Node* ptr = new Node (elem);
167     if (head == nullptr) {
168         head = ptr;
169         tail = ptr;
170         return;
171     } else {
172         tail -> next = ptr;
173         ptr -> prev = tail;
174         tail = ptr;
175     }
176
177     put_in_sorted_order(ptr);
178 }
179
180 template <typename T>
181 void FrankList<T>::pop_back() {
182     if (tail == nullptr) {
183         return;
184     }
185
186     if (tail -> prev != nullptr){
187         tail -> prev -> next = nullptr;

```

```

189     if (tail -> desc && tail -> asc){
190         tail -> asc -> desc = tail -> desc;
191         tail -> desc -> asc = tail -> asc;
192     }
193     Node* ptr = tail;
194     tail = tail -> prev;
195     delete ptr;
196 } else {
197     delete tail;
198     tail = nullptr;
199     head = nullptr;
200 }
201 }
202
203 template <typename T>
204 typename FrankList<T>::const_reference FrankList<T>::front() const {
205     if (head == nullptr){
206         throw std::out_of_range("Error");
207     }
208
209     return head -> val;
210 }
211
212 template <typename T>
213 typename FrankList<T>::reference FrankList<T>::front() {
214     if (head == nullptr){
215         throw std::out_of_range("Error");
216     }
217
218     return head -> val;
219 }
220
221 template <typename T>
222 typename FrankList<T>::const_reference FrankList<T>::back() const {
223     if (tail == nullptr){
224         throw std::out_of_range("Error");
225     }
226
227     return tail -> val;
228 }
229
230 template <typename T>
231 typename FrankList<T>::reference FrankList<T>::back() {
232     if (tail == nullptr){
233         throw std::out_of_range("Error");
234     }
235
236     return tail -> val;
237 }
238
239 template <typename T>
240 typename FrankList<T>::const_reference FrankList<T>::min() const {
241     if (ahead == nullptr){
242         throw std::out_of_range("Error");
243     }
244
245     return ahead -> val;
246 }
247

```



```

248 template <typename T>
249 typename FrankList<T>::reference FrankList<T>::min() {
250     if (ahead == nullptr){
251         throw std::out_of_range("Error");
252     }
253
254     return ahead -> val;
255 }
256
257 template <typename T>
258 typename FrankList<T>::const_reference FrankList<T>::max() const {
259     if (atail == nullptr){
260         throw std::out_of_range("Error");
261     }
262
263     return atail -> val;
264 }
265
266 template <typename T>
267 typename FrankList<T>::reference FrankList<T>::max() {
268     if (atail == nullptr){
269         throw std::out_of_range("Error");
270     }
271
272     return atail -> val;
273 }
274
275 template <typename T>
276 const FrankList<T>& FrankList<T>::operator=(const FrankList<T>& rhv) {
277     if (this != &rhv) {
278         clear();
279         for (auto it = rhv.cbegin(); it != rhv.cend(); ++it) {
280             push_back(*it);
281         }
282     }
283     return *this;
284 }
285
286 template <typename T>
287 const FrankList<T>& FrankList<T>::operator=(FrankList<T>&& rhv) {
288     if (this != &rhv){
289         clear();
290         head = rhv.head;
291         tail = rhv.tail;
292         ahead = rhv.ahead;
293         atail = rhv.atail;
294         rhv.head = nullptr;
295         rhv.tail = nullptr;
296         rhv.ahead = nullptr;
297         rhv.atail = nullptr;
298     }
299
300     return *this;
301 }
302
303 template <typename T>
304 const FrankList<T>& FrankList<T>::operator=(std::initializer_list<value_type> in
305     clear();
306     for(const auto& i : init){

```

```

307         push_back(i);
308     }
309
310     return *this;
311 }
312
313
314 template <typename T>
315 bool FrankList<T>::operator==(const FrankList<value_type>& rhv) const {
316     if (size() != rhv.size()){
317         return false;
318     }
319
320     auto i = cbegin();
321     auto j = rhv.cbegin();
322
323     while (i != cend() && j != rhv.cend()){
324         if(*i != *j){
325             return false;
326         }
327
328         ++i;
329         ++j;
330     }
331
332     return (i == cend() && j == rhv.cend());
333 }
334
335
336 template <typename T>
337 bool FrankList<T>::operator!=(const FrankList<value_type>& rhv) const {
338     return !(*this == rhv);
339 }
340
341 template <typename T>
342 bool FrankList<T>::operator<(const FrankList<value_type>& rhv) const {
343     size_type size1 = size();
344     size_type size2 = rhv.size();
345     if (size1 < size2){
346         return true;
347     } else if (size1 > size2){
348         return false;
349     }
350
351     auto i = cbegin();
352     auto j = rhv.cbegin();
353
354     while (i != cend() && j != rhv.cend()){
355         if (*i < *j){
356             return true;
357         } else if (*i > *j){
358             return false;
359         }
360
361         ++i;
362         ++j;
363     }
364
365     return !(i == cend() && j == rhv.cend());

```

```

366 }
367
368 template <typename T>
369 bool FrankList<T>::operator==(const FrankList<value_type>& rhv) const {
370     return (*this == rhv || *this < rhv);
371 }
372
373 template <typename T>
374 bool FrankList<T>::operator>(const FrankList<value_type>& rhv) const {
375     return !(*this <= rhv);
376 }
377
378 template <typename T>
379 bool FrankList<T>::operator>=(const FrankList<value_type>& rhv) const {
380     return (*this > rhv || *this == rhv);
381 }
382
383 template <typename T>
384 typename FrankList<T>::const_iterator FrankList<T>::cbegin() const {
385     return const_iterator(head);
386 }
387
388 template <typename T>
389 typename FrankList<T>::const_iterator FrankList<T>::cend() const {
390     return const_iterator(nullptr);
391 }
392
393 template <typename T>
394 typename FrankList<T>::const_reverse_iterator FrankList<T>::crbegin() const {
395     return const_reverse_iterator(tail);
396 }
397
398 template <typename T>
399 typename FrankList<T>::const_reverse_iterator FrankList<T>::crend() const {
400     return const_reverse_iterator(nullptr);
401 }
402
403 template <typename T>
404 typename FrankList<T>::const_asc_iterator FrankList<T>::cabegin() const {
405     return const_asc_iterator(ahead);
406 }
407
408 template <typename T>
409 typename FrankList<T>::const_asc_iterator FrankList<T>::caend() const {
410     return const_asc_iterator(nullptr);
411 }
412
413 template <typename T>
414 typename FrankList<T>::const_desc_iterator FrankList<T>::cdbegin() const {
415     return const_desc_iterator(atail);
416 }
417
418 template <typename T>
419 typename FrankList<T>::const_desc_iterator FrankList<T>::cdend() const {
420     return const_desc_iterator(nullptr);
421 }
422
423 template <typename T>
424 typename FrankList<T>::const_multi_iterator FrankList<T>::cmbegin() const {

```

```

425     return const_multi_iterator(head);
426 }
427
428 template <typename T>
429 typename FrankList<T>::const_multi_iterator FrankList<T>::cmend() const {
430     return const_multi_iterator(nullptr);
431 }
432
433 template <typename T>
434 typename FrankList<T>::const_multi_iterator FrankList<T>::cmabegin() const {
435     return const_multi_iterator(ahead);
436 }
437
438 template <typename T>
439 typename FrankList<T>::const_multi_iterator FrankList<T>::cmaend() const {
440     return const_multi_iterator(nullptr);
441 }
442
443 template <typename T>
444 typename FrankList<T>::const_multi_reverse_iterator FrankList<T>::cmrbegin() const {
445     return const_multi_reverse_iterator(tail);
446 }
447
448 template <typename T>
449 typename FrankList<T>::const_multi_reverse_iterator FrankList<T>::cmrend() const {
450     return const_multi_reverse_iterator(nullptr);
451 }
452
453 template <typename T>
454 typename FrankList<T>::const_multi_reverse_iterator FrankList<T>::cmrdbegin() const {
455     return const_multi_reverse_iterator(atail);
456 }
457
458 template <typename T>
459 typename FrankList<T>::const_multi_reverse_iterator FrankList<T>::cmrdend() const {
460     return const_multi_reverse_iterator(nullptr);
461 }
462
463 template <typename T>
464 typename FrankList<T>::iterator FrankList<T>::begin() {
465     return iterator(head);
466 }
467
468 template <typename T>
469 typename FrankList<T>::iterator FrankList<T>::end() {
470     return iterator(nullptr);
471 }
472
473 template <typename T>
474 typename FrankList<T>::reverse_iterator FrankList<T>::rbegin() {
475     return reverse_iterator(tail);
476 }
477
478 template <typename T>
479 typename FrankList<T>::reverse_iterator FrankList<T>::rend() {
480     return reverse_iterator(nullptr);
481 }
482
483 template <typename T>

```



```

484 typename FrankList<T>::asc_iterator FrankList<T>::abegin() {
485     return asc_iterator(ahead);
486 }
487
488 template <typename T>
489 typename FrankList<T>::asc_iterator FrankList<T>::aend() {
490     return asc_iterator(nullptr);
491 }
492
493 template <typename T>
494 typename FrankList<T>::desc_iterator FrankList<T>::dbegin() {
495     return desc_iterator(atail);
496 }
497
498 template <typename T>
499 typename FrankList<T>::desc_iterator FrankList<T>::dend() {
500     return desc_iterator(nullptr);
501 }
502
503 template <typename T>
504 typename FrankList<T>::multi_iterator FrankList<T>::mbegin() {
505     return multi_iterator(head);
506 }
507
508 template <typename T>
509 typename FrankList<T>::multi_iterator FrankList<T>::mend() {
510     return multi_iterator(nullptr);
511 }
512
513 template <typename T>
514 typename FrankList<T>::multi_iterator FrankList<T>::mabegin() {
515     return multi_iterator(ahead);
516 }
517
518 template <typename T>
519 typename FrankList<T>::multi_iterator FrankList<T>::maend() {
520     return multi_iterator(nullptr);
521 }
522
523 template <typename T>
524 typename FrankList<T>::multi_reverse_iterator FrankList<T>::mrbegin() {
525     return multi_reverse_iterator(tail);
526 }
527
528 template <typename T>
529 typename FrankList<T>::multi_reverse_iterator FrankList<T>::mrend() {
530     return multi_reverse_iterator(nullptr);
531 }
532
533 template <typename T>
534 typename FrankList<T>::multi_reverse_iterator FrankList<T>::mrdbegin() {
535     return multi_reverse_iterator(atail);
536 }
537
538 template <typename T>
539 typename FrankList<T>::multi_reverse_iterator FrankList<T>::mrdend() {
540     return multi_reverse_iterator(nullptr);
541 }
542

```

```

543
544 template <typename T>
545 template <typename iter>
546 iter FrankList<T>::insert(iter pos, size_type size, const_reference val){
547     if (empty() && pos != begin()){
548         throw std::invalid_argument("Error");
549     }
550
551     if (pos == begin()){
552         for(size_type s = 0; s < size; ++s){
553             push_front(val);
554         }
555     } else if (pos == end()){
556         for(size_type s = 0; s < size; ++s){
557             push_back(val);
558         }
559     } else {
560         for(size_type s = 0; s < size; ++s){
561             pos = insert_def(pos, val);
562         }
563     }
564
565     return pos;
566 }
567
568 template <typename T>
569 template <typename iter>
570 iter FrankList<T>::insert(iter pos, std::initializer_list<value_type> init){
571     if (empty() && pos != begin()){
572         throw std::invalid_argument("Error");
573     }
574
575     if (pos == begin()){
576         for(const auto& i : init){
577             push_front(i);
578         }
579     } else if (pos == end()){
580         for(const auto& i : init){
581             push_back(i);
582         }
583     } else {
584         for(const auto& i : init){
585             pos = insert_def(pos, i);
586         }
587     }
588
589     return pos;
590 }
591
592 template <typename T>
593 template <typename iter, typename input_iterator>
594 iter FrankList<T>::insert(iter pos, input_iterator f, input_iterator l){
595     if (empty() && pos != begin()){
596         throw std::invalid_argument("Error");
597     }
598
599     if (f == l){
600         return pos;
601     }

```

```

602
603     if (pos == begin()){
604         for(input_iterator it = f; it != l; ++it){
605             push_front(*it);
606         }
607     } else if (pos == end()){
608         for(input_iterator it = f; it != l; ++it){
609             push_back(*it);
610         }
611     } else {
612         for(input_iterator it = f; it != l; ++it){
613             pos = insert_def(pos, *it);
614         }
615     }
616
617     return pos;
618 }
619
620 template <typename T>
621 template <typename iter>
622 iter FrankList<T>::erase(iter pos){
623     if (empty()){
624         throw std::invalid_argument("Error");
625     }
626
627     Node* ptr = pos.ptr;
628     iter next(pos);
629     ++next;
630
631     if (pos == begin()) {
632         pop_front();
633         return next;
634     } else if (pos == iter(tail)) {
635         pop_back();
636         return next;
637     }
638
639     if (ptr->prev && ptr->next) {
640         ptr->prev->next = ptr->next;
641         ptr->next->prev = ptr->prev;
642     }
643
644     if (ptr->desc && ptr->asc) {
645         ptr->desc->asc = ptr->asc;
646         ptr->asc->desc = ptr->desc;
647     }
648
649     delete ptr;
650     return next;
651 }
652
653
654 template <typename T>
655 template <typename iter>
656 iter FrankList<T>::erase(iter f, iter l){
657     iter it = f;
658     while (it != l){
659         it = erase(it);
660     }

```

```

661
662     return it;
663 }
664
665 template <typename T>
666 typename FrankList<T>::size_type FrankList<T>::remove(const_reference val){
667     size_type count = 0;
668     iterator it = begin();
669     while (it != end()){
670         if (it != end() && *it == val){
671             std::cout << *it << std::endl;
672             it = erase(it);
673             ++count;
674             if (it == end()){
675                 break;
676             }
677         } else {
678             ++it;
679         }
680     }
681
682     return count;
683 }
684
685 template <typename T>
686 template <typename unary_predicate>
687 typename FrankList<T>::size_type FrankList<T>::remove_if(unary_predicate func){
688     size_type count = 0;
689     iterator it = begin();
690     while (it != end()){
691         if(func(*it)){
692             it = erase(it);
693             ++count;
694         } else {
695             ++it;
696         }
697     }
698
699     return count;
700 }
701
702 template <typename T>
703 void FrankList<T>::reverse() {
704     std::swap(head, tail);
705     Node* it = head;
706     while (it != nullptr) {
707         std::swap(it->next, it->prev);
708         it = it->next;
709     }
710 }
711
712 template <typename T>
713 void FrankList<T>::sort(bool reversed) {
714     if (!ahead) {
715         return;
716     }
717     if (!reversed){
718         Node* tmp = ahead;
719         while (tmp != nullptr){

```

```

720     tmp -> next = tmp -> asc;
721     tmp -> prev = tmp -> desc;
722     tmp = tmp -> next;
723 }
724
725 head = ahead;
726 tail = atail;
727 } else {
728     Node* tmp = atail;
729     while (tmp != nullptr){
730         tmp -> next = tmp -> desc;
731         tmp -> prev = tmp -> asc;
732         tmp = tmp -> next;
733     }
734
735     head = atail;
736     tail = ahead;
737 }
738 }
739
740 template <typename T>
741 typename FrankList<T>::iterator FrankList<T>::find(const_reference elem){
742     iterator it = begin();
743     while (it != end()){
744         if (*it != elem){
745             ++it;
746         } else {
747             break;
748         }
749     }
750
751     organize_left(it.ptr);
752     return it;
753 }
754
755 template <typename T>
756 typename FrankList<T>::iterator FrankList<T>::rfind(const_reference elem){
757     const_reverse_iterator it = crbegin();
758     while (it != crend()){
759         if(*it != elem){
760             ++it;
761         } else {
762             break;
763         }
764     }
765
766     organize_right(it.ptr);
767     return it;
768 }
769
770
771
772 template <typename T>
773 template <typename unary_predicate>
774 void FrankList<T>::traverse(unary_predicate func, bool sorted, bool reversed){
775     if (!head) {
776         return;
777     }
778     if (!sorted && !reversed) {

```

```

779         for (auto it = begin(); it != end(); ++it) {
780             func(*it);
781         }
782     }
783     else if (!sorted) {
784         for (auto it = rbegin(); it != rend(); ++it) {
785             func(*it);
786         }
787     }
788     else if (!reversed) {
789         for (auto it = abegin(); it != aend(); ++it) {
790             func(*it);
791         }
792     }
793     else {
794         for (auto it = dbegin(); it != dend(); ++it) {
795             func(*it);
796         }
797     }
798 }
799
800 template <typename T>
801 void FrankList<T>::print(bool sorted, bool reversed) {
802     if(!head){
803         std::cout << std::endl;
804     }
805
806     if (!sorted && !reversed) {
807         for (auto it = begin(); it != end(); ++it) {
808             std::cout << *it << ' ';
809         }
810     }
811     else if (!sorted) {
812         for (auto it = rbegin(); it != rend(); ++it) {
813             std::cout << *it << ' ';
814         }
815     }
816     else if (!reversed) {
817         for (auto it = abegin(); it != aend(); ++it) {
818             std::cout << *it << ' ';
819         }
820     }
821     else {
822         for (auto it = dbegin(); it != dend(); ++it) {
823             std::cout << *it << ' ';
824         }
825     }
826 }
827
828
829 template <typename T>
830 void FrankList<T>::put_in_sorted_order(Node* ptr) {
831     if (!ahead){
832         ahead = ptr;
833         atail = ptr;
834         return;
835     }
836
837     Node* tmp = ahead;

```

```

838 while (tmp -> asc && tmp -> val <= ptr -> val){
839     tmp = tmp -> asc;
840 }
841
842 if (tmp && tmp->val < ptr->val) {
843     ptr->asc = tmp->asc;
844     if (tmp->asc) {
845         tmp->asc->desc = ptr;
846     } else {
847         atail = ptr;
848     }
849     tmp->asc = ptr;
850     ptr->desc = tmp;
851 } else {
852     ptr->desc = tmp->desc;
853     if (tmp->desc) {
854         tmp->desc->asc = ptr;
855     } else {
856         ahead = ptr;
857     }
858     ptr->asc = tmp;
859     tmp->desc = ptr;
860 }
861 }
862
863
864 template <typename T>
865 void FrankList<T>::organize_left(Node* ptr) {
866     if (ptr == nullptr || ptr == head){
867         return;
868     }
869
870     Node* node = ptr -> prev;
871     node -> prev -> next = ptr;
872     ptr -> prev = node -> prev;
873     node -> prev = ptr;
874     node -> next = ptr -> next;
875     node -> next -> prev = node;
876     ptr -> next = node;
877
878     if (node == head){
879         head = ptr;
880     }
881
882     if (ptr == tail){
883         tail = node;
884     }
885 }
886
887
888 template <typename T>
889 void FrankList<T>::organize_right(Node* ptr) {
890     if (ptr == nullptr || ptr == tail){
891         return;
892     }
893
894     Node* nodeP = ptr -> prev;

```

```

897     Node* nodeN = ptr -> next;
898
899     nodeP -> next = nodeN;
900     nodeN -> prev = nodeP;
901     ptr -> next = nodeN -> next;
902     ptr -> prev = nodeN;
903     nodeN -> next -> prev = ptr;
904     nodeN -> next = ptr;
905
906     if (ptr == head){
907         head = nodeN;
908     }
909
910     if (nodeN == tail){
911         tail = ptr;
912     }
913 }
914
915 template <typename T>
916 template <typename iter>
917 iter FrankList<T>::insert_def(iter pos, const_reference val) {
918     Node* ptr = new Node(val);
919     Node* node = pos.ptr;
920
921     if (node == nullptr) {
922         if (!tail) {
923             head = tail = ptr;
924         } else {
925             tail->next = ptr;
926             ptr->prev = tail;
927             tail = ptr;
928         }
929     } else {
930         ptr->next = node;
931         ptr->prev = node->prev;
932         if (node->prev) {
933             node->prev->next = ptr;
934         } else {
935             head = ptr;
936         }
937         node->prev = ptr;
938     }
939     put_in_sorted_order(ptr);
940     return iter(ptr);
941 }
942
943 template <typename T>
944 template <typename iter>
945 iter FrankList<T>::insert_rev(iter pos, const_reference val) {
946     Node* ptr = new Node(val);
947     Node* node = pos.ptr;
948
949     if (node == nullptr) {
950         if (!tail) {
951             head = tail = ptr;
952         } else {
953             tail->next = ptr;
954             ptr->prev = tail;
955             tail = ptr;

```

```

956     }
957 } else {
958     ptr->prev = node;
959     ptr->next = node->next;
960     if (node->next) {
961         node->next->prev = ptr;
962     } else {
963         tail = ptr;
964     }
965     node->next = ptr;
966 }
967 put_in_sorted_order(ptr);
968 return iter(ptr);
969 }
970
971
972
973 ///////////////////////////////////////////////////////////////////.
974
975 template <typename T>
976 FrankList<T>::base_iterator::~~base_iterator() {
977     ptr = nullptr;
978 }
979
980 template <typename T>
981 bool FrankList<T>::base_iterator::operator==(const base_iterator& rhv) const {
982     return this -> ptr == rhv.ptr;
983 }
984
985 template <typename T>
986 bool FrankList<T>::base_iterator::operator!=(const base_iterator& rhv) const {
987     return !(this->ptr == rhv.ptr);
988 }
989
990 template <typename T>
991 FrankList<T>::base_iterator::base_iterator(Node* ptr) : ptr(ptr) {}
992
993 template <typename T>
994 FrankList<T>::const_iterator::const_iterator(const base_iterator& rhv) : base_iterator(rhv.ptr) {}
995
996 template <typename T>
997 FrankList<T>::const_iterator::const_iterator(base_iterator&& rhv) : base_iterator(rhv.ptr) {}
998     rhv.ptr = nullptr;
999 }
1000
1001 template <typename T>
1002 const typename FrankList<T>::const_iterator& FrankList<T>::const_iterator::operator=(const base_iterator& rhv) {
1003     this -> ptr = rhv.ptr;
1004     return *this;
1005 }
1006
1007 template <typename T>
1008 const typename FrankList<T>::const_iterator& FrankList<T>::const_iterator::operator=(const base_iterator&& rhv) {
1009     if (this == &rhv){
1010         return *this;
1011     }
1012     this -> ptr = rhv.ptr;
1013     rhv.ptr = nullptr;
1014 }

```

```

1015     return *this;
1016 }
1017
1018 template <typename T>
1019 typename FrankList<T>::const_reference FrankList<T>::const_iterator::operator*() const {
1020     return this -> ptr -> val;
1021 }
1022
1023 template <typename T>
1024 typename FrankList<T>::const_pointer FrankList<T>::const_iterator::operator->() const {
1025     return this -> ptr;
1026 }
1027
1028 template <typename T>
1029 const typename FrankList<T>::const_iterator& FrankList<T>::const_iterator::operator=(const base_iterator& rhv) {
1030     this -> ptr = this -> ptr -> next;
1031     return *this;
1032 }
1033
1034 template <typename T>
1035 const typename FrankList<T>::const_iterator FrankList<T>::const_iterator::operator++() const {
1036     const_iterator tmp = *this;
1037     ++(*this);
1038     return tmp;
1039 }
1040
1041 template <typename T>
1042 const typename FrankList<T>::const_iterator& FrankList<T>::const_iterator::operator--() const {
1043     this -> ptr = this -> ptr -> prev;
1044     return *this;
1045 }
1046
1047 template <typename T>
1048 const typename FrankList<T>::const_iterator FrankList<T>::const_iterator::operator++(int) const {
1049     const_iterator tmp = *this;
1050     --(*this);
1051     return tmp;
1052 }
1053
1054 template <typename T>
1055 FrankList<T>::const_iterator::const_iterator(Node* ptr) : base_iterator(ptr) {}
1056
1057 template <typename T>
1058 FrankList<T>::iterator::iterator(const base_iterator& rhv) : const_iterator(rhv.ptr) {}
1059
1060 template <typename T>
1061 FrankList<T>::iterator::iterator(base_iterator&& rhv) : const_iterator(rhv.ptr) {}
1062     rhv.ptr = nullptr;
1063 }
1064
1065 template <typename T>
1066 typename FrankList<T>::reference FrankList<T>::iterator::operator*() const {
1067     return this -> ptr -> val;
1068 }
1069
1070 template <typename T>
1071 typename FrankList<T>::pointer FrankList<T>::iterator::operator->() const {
1072     return this -> ptr;
1073 }

```

```

1074
1075 template <typename T>
1076 const typename FrankList<T>::iterator& FrankList<T>::iterator::operator=(const b
1077     this -> ptr = rhv.ptr;
1078     return *this;
1079 }
1080
1081 template <typename T>
1082 const typename FrankList<T>::iterator& FrankList<T>::iterator::operator=(base_it
1083     if (this == &rhv){
1084         return *this;
1085     }
1086
1087     this -> ptr = rhv.ptr;
1088     rhv.ptr = nullptr;
1089     return *this;
1090 }
1091
1092 template <typename T>
1093 FrankList<T>::iterator::iterator(Node* ptr) : const_iterator(ptr) {}
1094
1095 template <typename T>
1096 FrankList<T>::const_reverse_iterator::const_reverse_iterator(const base_iterator&
1097
1098 template <typename T>
1099 FrankList<T>::const_reverse_iterator::const_reverse_iterator(base_iterator&& rhv
1100     rhv.ptr = nullptr;
1101 }
1102
1103 template <typename T>
1104 const typename FrankList<T>::const_reverse_iterator& FrankList<T>::const_reverse
1105 base_iterator& rhv){
1106     this -> ptr = rhv.ptr;
1107     return *this;
1108 }
1109
1110 template <typename T>
1111 const typename FrankList<T>::const_reverse_iterator& FrankList<T>::const_reverse
1112 (base_iterator&& rhv) {
1113     if (this = &rhv){
1114         return *this;
1115     }
1116
1117     this -> ptr = rhv.ptr;
1118     rhv.ptr = nullptr;
1119     return *this;
1120 }
1121
1122 template <typename T>
1123 typename FrankList<T>::const_reference FrankList<T>::const_reverse_iterator::ope
1124     return this->ptr->val;
1125 }
1126
1127 template <typename T>
1128 typename FrankList<T>::const_pointer FrankList<T>::const_reverse_iterator::opera
1129     return this->ptr;
1130 }
1131
1132 template <typename T>

```

```

1131 const typename FrankList<T>::const_reverse_iterator& FrankList<T>::const_reverse
1132     this->ptr = this->ptr->prev;
1133     return *this;
1134 }
1135
1136 template <typename T>
1137 const typename FrankList<T>::const_reverse_iterator FrankList<T>::const_reverse_
1138     const_reverse_iterator tmp = *this;
1139     ++(*this);
1140     return tmp;
1141 }
1142
1143 template <typename T>
1144 const typename FrankList<T>::const_reverse_iterator& FrankList<T>::const_reverse
1145     this->ptr = this->ptr->next;
1146     return *this;
1147 }
1148
1149 template <typename T>
1150 const typename FrankList<T>::const_reverse_iterator FrankList<T>::const_reverse_
1151     const_reverse_iterator tmp = *this;
1152     --(*this);
1153     return tmp;
1154 }
1155
1156 template <typename T>
1157 FrankList<T>::const_reverse_iterator::const_reverse_iterator(Node* ptr) : base_i
1158
1159 template <typename T>
1160 FrankList<T>::reverse_iterator::reverse_iterator(const base_iterator& rhv) : con
1161
1162 template <typename T>
1163 FrankList<T>::reverse_iterator::reverse_iterator(base_iterator&& rhv) : const_re
1164     rhv.ptr = nullptr;
1165 }
1166
1167 template <typename T>
1168 typename FrankList<T>::reference FrankList<T>::reverse_iterator::operator*() {
1169     return this->ptr->val;
1170 }
1171
1172 template <typename T>
1173 typename FrankList<T>::pointer FrankList<T>::reverse_iterator::operator->() {
1174     return this->ptr;
1175 }
1176
1177 template <typename T>
1178 const typename FrankList<T>::reverse_iterator& FrankList<T>::reverse_iterator::o
1179 {
1180     this -> ptr = rhv.ptr;
1181     return *this;
1182 }
1183
1184 template <typename T>
1185 const typename FrankList<T>::reverse_iterator& FrankList<T>::reverse_iterator::o
1186     if (this = &rhv){
1187         return *this;
1188     }

```



```

1189     }
1190
1191     this -> ptr = rhv.ptr;
1192     rhv.ptr = nullptr;
1193     return *this;
1194 }
1195
1196 template <typename T>
1197 FrankList<T>::reverse_iterator::reverse_iterator(Node* ptr) : const_reverse_itera
1198
1199 template <typename T>
1200 FrankList<T>::const_asc_iterator::const_asc_iterator(const base_iterator& rhv) :
1201
1202 template <typename T>
1203 FrankList<T>::const_asc_iterator::const_asc_iterator(base_iterator&& rhv) : base_
1204     rhv.ptr = nullptr;
1205 }
1206
1207 template <typename T>
1208 const typename FrankList<T>::const_asc_iterator& FrankList<T>::const_asc_iterato
1209     rhv){
1210     this -> ptr = rhv.ptr;
1211     return *this;
1212 }
1213
1214 template <typename T>
1215 const typename FrankList<T>::const_asc_iterator& FrankList<T>::const_asc_iterato
1216 {
1217     if (this = &rhv){
1218         return *this;
1219     }
1220
1221     this -> ptr = rhv.ptr;
1222     rhv.ptr = nullptr;
1223     return *this;
1224 }
1225
1226 template <typename T>
1227 typename FrankList<T>::const_reference FrankList<T>::const_asc_iterator::operator
1228     return this->ptr->val;
1229 }
1230
1231 template <typename T>
1232 typename FrankList<T>::const_pointer FrankList<T>::const_asc_iterator::operator-:
1233     return this->ptr;
1234 }
1235
1236 template <typename T>
1237 const typename FrankList<T>::const_asc_iterator& FrankList<T>::const_asc_iterato
1238     this -> ptr = this -> ptr -> asc;
1239     return *this;
1240 }
1241
1242 template <typename T>
1243 const typename FrankList<T>::const_asc_iterator FrankList<T>::const_asc_iterator
1244     const_asc_iterator tmp = *this;
1245     ++(*this);
1246     return tmp;
1247 }

```

```

1246
1247 template <typename T>
1248 const typename FrankList<T>::const_asc_iterator& FrankList<T>::const_asc_iterato
1249     this -> ptr = this -> ptr -> desc;
1250     return *this;
1251 }
1252
1253 template <typename T>
1254 const typename FrankList<T>::const_asc_iterator FrankList<T>::const_asc_iterator
1255     const_asc_iterator tmp = *this;
1256     --(*this);
1257     return tmp;
1258 }
1259
1260 template <typename T>
1261 FrankList<T>::const_asc_iterator::const_asc_iterator(Node* ptr) : base_iterator(|
1262
1263 template <typename T>
1264 FrankList<T>::asc_iterator::asc_iterator(const base_iterator& rhv) : const_asc_i
1265     // this -> ptr = rhv.ptr;
1266 }
1267
1268 template <typename T>
1269 FrankList<T>::asc_iterator::asc_iterator(base_iterator&& rhv) : const_asc_iterat
1270     rhv.ptr = nullptr;
1271 }
1272
1273 template <typename T>
1274 typename FrankList<T>::reference FrankList<T>::asc_iterator::operator*() {
1275     return this->ptr->val;
1276 }
1277
1278 template <typename T>
1279 typename FrankList<T>::pointer FrankList<T>::asc_iterator::operator->() {
1280     return this->ptr;
1281 }
1282
1283 template <typename T>
1284 const typename FrankList<T>::asc_iterator& FrankList<T>::asc_iterator::operator=
1285     this -> ptr = rhv.ptr;
1286     return *this;
1287 }
1288
1289 template <typename T>
1290 const typename FrankList<T>::asc_iterator& FrankList<T>::asc_iterator::operator=
1291     if (this = &rhv){
1292         return *this;
1293     }
1294
1295     this -> ptr = rhv.ptr;
1296     rhv.ptr = nullptr;
1297     return *this;
1298 }
1299
1300 template <typename T>
1301 FrankList<T>::asc_iterator::asc_iterator(Node* ptr) : const_asc_iterator (ptr)
1302
1303 template <typename T>
1304 FrankList<T>::const_desc_iterator::const_desc_iterator(const base_iterator& rhv)

```

```

1305
1306 template <typename T>
1307 FrankList<T>::const_desc_iterator::const_desc_iterator(base_iterator&& rhv) : ba
1308     rhv.ptr = nullptr;
1309 }
1310
1311 template <typename T>
1312 const typename FrankList<T>::const_desc_iterator& FrankList<T>::const_desc_itera
1313     base_iterator& rhv){
1314     this -> ptr = rhv.ptr;
1315     return *this;
1316 }
1317
1318 template <typename T>
1319 const typename FrankList<T>::const_desc_iterator& FrankList<T>::const_desc_itera
1320     rhv) {
1321     this -> ptr = rhv.ptr;
1322     rhv.ptr = nullptr;
1323     return *this;
1324 }
1325
1326 template <typename T>
1327 typename FrankList<T>::const_reference FrankList<T>::const_desc_iterator::operat
1328     return this -> ptr -> val;
1329 }
1330
1331 template <typename T>
1332 typename FrankList<T>::const_pointer FrankList<T>::const_desc_iterator::operator
1333     return this -> ptr;
1334 }
1335
1336 template <typename T>
1337 const typename FrankList<T>::const_desc_iterator& FrankList<T>::const_desc_itera
1338     this -> ptr = this -> ptr -> desc;
1339     return *this;
1340 }
1341
1342 template <typename T>
1343 const typename FrankList<T>::const_desc_iterator FrankList<T>::const_desc_iterat
1344     const_desc_iterator tmp = *this;
1345     ++(*this);
1346     return tmp;
1347 }
1348
1349 template <typename T>
1350 const typename FrankList<T>::const_desc_iterator& FrankList<T>::const_desc_itera
1351     this -> ptr = this -> ptr -> asc;
1352     return *this;
1353 }
1354
1355 template <typename T>
1356 const typename FrankList<T>::const_desc_iterator FrankList<T>::const_desc_iterat
1357     const_desc_iterator tmp = *this;
1358     --(*this);
1359     return tmp;
1360 }
1361
1362 template <typename T>
1363 FrankList<T>::const_desc_iterator::const_desc_iterator(Node* ptr) : base_iterato

```

```

1362
1363 template <typename T>
1364 FrankList<T>::desc_iterator::desc_iterator(const base_iterator& rhv) : const_des
1365
1366 template <typename T>
1367 FrankList<T>::desc_iterator::desc_iterator(base_iterator&& rhv) : const_desc_ite
1368     rhv.ptr = nullptr;
1369 }
1370
1371 template <typename T>
1372 typename FrankList<T>::reference FrankList<T>::desc_iterator::operator*() {
1373     return this -> ptr -> val;
1374 }
1375
1376 template <typename T>
1377 typename FrankList<T>::pointer FrankList<T>::desc_iterator::operator->() {
1378     return this -> ptr;
1379 }
1380
1381 template <typename T>
1382 const typename FrankList<T>::desc_iterator& FrankList<T>::desc_iterator::operato
1383     this -> ptr = rhv.ptr;
1384     return *this;
1385 }
1386
1387 template <typename T>
1388 const typename FrankList<T>::desc_iterator& FrankList<T>::desc_iterator::operato
1389     this -> ptr = rhv.ptr;
1390     rhv.ptr = nullptr;
1391     return *this;
1392 }
1393
1394 template <typename T>
1395 FrankList<T>::desc_iterator::desc_iterator(Node* ptr) : const_desc_iterator(ptr)
1396
1397 template <typename T>
1398 FrankList<T>::const_multi_iterator::const_multi_iterator(const base_iterator& rh
1399
1400 template <typename T>
1401 FrankList<T>::const_multi_iterator::const_multi_iterator(base_iterator&& rhv) :
1402     rhv.ptr = nullptr;
1403 }
1404
1405 template <typename T>
1406 const typename FrankList<T>::const_multi_iterator& FrankList<T>::const_multi_ite
1407     base_iterator& rhv){
1408     this -> ptr = rhv.ptr;
1409     return *this;
1410 }
1411
1412 template <typename T>
1413 const typename FrankList<T>::const_multi_iterator& FrankList<T>::const_multi_ite
1414     rhv) {
1415     this -> ptr = rhv.ptr;
1416     rhv.ptr = nullptr;
1417     return *this;
1418 }
1419
1420 template <typename T>

```

```

1419 typename FrankList<T>::const_reference FrankList<T>::const_multi_iterator::opera
1420     return this -> ptr -> val;
1421 }
1422
1423 template <typename T>
1424 typename FrankList<T>::const_pointer FrankList<T>::const_multi_iterator::operato
1425     return this -> ptr;
1426 }
1427
1428 template <typename T>
1429 const typename FrankList<T>::const_multi_iterator& FrankList<T>::const_multi_ite
1430     if (mode){
1431         this -> ptr = this -> ptr -> next;
1432     } else {
1433         this -> ptr = this -> ptr -> asc;
1434     }
1435
1436     return *this;
1437 }
1438
1439 template <typename T>
1440 const typename FrankList<T>::const_multi_iterator FrankList<T>::const_multi_iter
1441     const_multi_iterator tmp(*this);
1442     if (mode){
1443         this -> ptr = this -> ptr -> next;
1444     } else {
1445         this -> ptr = this -> ptr -> asc;
1446     }
1447
1448     return tmp;
1449 }
1450
1451 template <typename T>
1452 const typename FrankList<T>::const_multi_iterator& FrankList<T>::const_multi_ite
1453     if (mode){
1454         this -> ptr = this -> ptr -> prev;
1455     } else {
1456         this -> ptr = this -> ptr -> desc;
1457     }
1458
1459     return *this;
1460 }
1461
1462 template <typename T>
1463 const typename FrankList<T>::const_multi_iterator FrankList<T>::const_multi_iter
1464     const_multi_iterator tmp(*this);
1465     if (mode){
1466         this -> ptr = this -> ptr -> prev;
1467     } else {
1468         this -> ptr = this -> ptr -> desc;
1469     }
1470
1471     return tmp;
1472 }
1473
1474 template <typename T>
1475 void FrankList<T>::const_multi_iterator::chmod() {
1476     mode = !mode;
1477 }

```

```

1478
1479 template <typename T>
1480 FrankList<T>::const_multi_iterator::const_multi_iterator(Node* ptr) : base_itera
1481
1482 template <typename T>
1483 FrankList<T>::multi_iterator::multi_iterator(const base_iterator& rhv) : const_m
1484
1485 template <typename T>
1486 FrankList<T>::multi_iterator::multi_iterator(base_iterator&& rhv) : const_multi_
1487     rhv.ptr = nullptr;
1488 }
1489
1490 template <typename T>
1491 typename FrankList<T>::reference FrankList<T>::multi_iterator::operator*() {
1492     return this -> ptr -> val;
1493 }
1494
1495 template <typename T>
1496 typename FrankList<T>::pointer FrankList<T>::multi_iterator::operator->() {
1497     return this -> ptr;
1498 }
1499
1500 template <typename T>
1501 const typename FrankList<T>::multi_iterator& FrankList<T>::multi_iterator::opera
1502     this -> ptr = rhv.ptr;
1503     return *this;
1504 }
1505
1506 template <typename T>
1507 const typename FrankList<T>::multi_iterator& FrankList<T>::multi_iterator::opera
1508     this -> ptr = rhv.ptr;
1509     rhv.ptr = nullptr;
1510     return *this;
1511 }
1512
1513 template <typename T>
1514 FrankList<T>::multi_iterator::multi_iterator(Node* ptr) : const_multi_iterator(p
1515
1516 template <typename T>
1517 FrankList<T>::const_multi_reverse_iterator::const_multi_reverse_iterator(const b
1518     base_iterator(rhv.ptr) {}
1519
1520 template <typename T>
1521 FrankList<T>::const_multi_reverse_iterator::const_multi_reverse_iterator(base_it
1522     base_iterator(rhv.ptr) {
1523         rhv.ptr = nullptr;
1524     }
1525
1526 template <typename T>
1527 const typename FrankList<T>::const_multi_reverse_iterator& FrankList<T>::const_m
1528     (const base_iterator& rhv){
1529         this -> ptr = rhv.ptr;
1530         return *this;
1531     }
1532
1533 template <typename T>
1534 const typename FrankList<T>::const_multi_reverse_iterator& FrankList<T>::const_m
1535     (base_iterator&& rhv) {
1536         this -> ptr = rhv.ptr;
1537         rhv.ptr = nullptr;

```

```

1534     return *this;
1535 }
1536
1537 template <typename T>
1538 typename FrankList<T>::const_reference FrankList<T>::const_multi_reverse_iterato
1539     return this -> ptr -> val;
1540 }
1541
1542 template <typename T>
1543 typename FrankList<T>::const_pointer FrankList<T>::const_multi_reverse_iterator:
1544     return this -> ptr;
1545 }
1546
1547 template <typename T>
1548 const typename FrankList<T>::const_multi_reverse_iterator& FrankList<T>::const_m
1549 (){
1550     if (mode){
1551         this -> ptr = this -> ptr -> prev;
1552     } else {
1553         this -> ptr = this -> ptr -> desc;
1554     }
1555     return *this;
1556 }
1557
1558 template <typename T>
1559 const typename FrankList<T>::const_multi_reverse_iterator FrankList<T>::const_mu
1560 (int){
1561     const_multi_reverse_iterator tmp(*this);
1562     ++(*this);
1563     return tmp;
1564 }
1565
1566 template <typename T>
1567 const typename FrankList<T>::const_multi_reverse_iterator& FrankList<T>::const_m
1568 (){
1569     if (mode){
1570         this -> ptr = this -> ptr -> next;
1571     } else {
1572         this -> ptr = this -> ptr -> asc;
1573     }
1574     return *this;
1575 }
1576
1577 template <typename T>
1578 const typename FrankList<T>::const_multi_reverse_iterator FrankList<T>::const_mu
1579 (int){
1580     const_multi_reverse_iterator tmp(*this);
1581     --(*this);
1582     return tmp;
1583 }
1584
1585 template <typename T>
1586 void FrankList<T>::const_multi_reverse_iterator::chmod() {
1587     mode = !mode;
1588 }
1589
1590 template <typename T>
1591 FrankList<T>::const_multi_reverse_iterator::const_multi_reverse_iterator(Node* p

```

```

1590
1591 template <typename T>
1592 FrankList<T>::multi_reverse_iterator::multi_reverse_iterator(const base_iterator&
1593     const_multi_reverse_iterator(rhv.ptr) {}
1594
1595 template <typename T>
1596 FrankList<T>::multi_reverse_iterator::multi_reverse_iterator(base_iterator&& rhv
1597     const_multi_reverse_iterator(rhv.ptr) {
1598     rhv.ptr = nullptr;
1599 }
1600
1601 template <typename T>
1602 typename FrankList<T>::reference FrankList<T>::multi_reverse_iterator::operator*
1603     return this -> ptr -> val;
1604 }
1605
1606 template <typename T>
1607 typename FrankList<T>::pointer FrankList<T>::multi_reverse_iterator::operator->()
1608     return this -> ptr;
1609 }
1610
1611 template <typename T>
1612 const typename FrankList<T>::multi_reverse_iterator& FrankList<T>::multi_reverse
1613     base_iterator& rhv){
1614     this -> ptr = rhv.ptr;
1615     return *this;
1616 }
1617
1618 template <typename T>
1619 const typename FrankList<T>::multi_reverse_iterator& FrankList<T>::multi_reverse
1620     (base_iterator&& rhv) {
1621     this -> ptr = rhv.ptr;
1622     rhv.ptr = nullptr;
1623     return *this;
1624 }
1625
1626 template <typename T>
1627 FrankList<T>::multi_reverse_iterator::multi_reverse_iterator(Node* ptr) : const_
1628 }
1629
1630 #endif
1631
1632
1633
1634
1635

```