

The Core Implementations of The OpenGC³ Project

Dong Nai-Jia

November 13, 2017

License

This Project Is Distributed Under The MIT License.

Abstract

OpenGC³ is a collection consisting of a few fundamental containers (and adaptors) for programming in C language. Most of the functions are implemented as C preprocessor function-like macros strictly conforming to the C99 standards, and therefore they outperform many other libraries, since they're expanded at compile time and aggressively optimized by compilers.

Contents

1	Source Code	2
1.1	list	2
1.1.1	list/base.h	2
1.1.2	list/ccxll.h	10
1.1.3	list/ccdll.h	24
1.1.4	list/ccsll.h	36
1.1.5	list/extd-base.h	46
1.1.6	list/extd-ccxll.h	48
1.1.7	list/extd-ccdll.h	51
1.1.8	list/extd-ccsll.h	53
1.2	base	55
1.2.1	base/mesg.h	55
1.2.2	base/misc.h	56
1.2.3	base/pool.h	58
1.2.4	base/snym.h	64

1 Source Code

1.1 list

1.1.1 list/base.h

```
1 #ifndef OPENG3_LIST_BASE_H
2 #define OPENG3_LIST_BASE_H
3
4
5 /* exclusive or */
6
7
8 #define XOR2(_addr_a, _addr_b) \
9 ( \
10     (void*)((uintptr_t)(void*)(_addr_a) ^ \
11         (uintptr_t)(void*)(_addr_b)) \
12 )
13
14
15 #define XOR3(_addr_a, _addr_b, _addr_c) \
16 ( \
17     (void*)((uintptr_t)(void*)(_addr_a) ^ \
18         (uintptr_t)(void*)(_addr_b) ^ \
19         (uintptr_t)(void*)(_addr_c)) \
20 )
21
22
23 #define XOR2_SWAP(_addr_a, _addr_b) \
24 \
25 VOID_EXPR_ \
26 ( \
27     (_addr_a) = XOR2((_addr_a), (_addr_b)), \
28     (_addr_b) = XOR2((_addr_a), (_addr_b)), \
29     (_addr_a) = XOR2((_addr_a), (_addr_b)) \
30 )
31
32
33
34 /* cc_ll initialize */
35
36
37 #define cc_ll_init_extd(cc_ll, _start, _ratio, _thrsh, _ll_) \
38 \
39 STATEMENT_ \
40 ( \
41     (_cc_ll) = NULL; \
42 \
43     _##_ll_##_init_extd((_cc_ll), (_start), (_ratio), (_thrsh), 1); \
44 \
45     _itarr_init((_cc_ll), _ll_); \
```

```

46     _##_ll_##_iter_init((_cc_ll)->_iter, (_cc_ll), 1);                                \
47 )                                                                                          \
48                                                                                          \
49                                                                                          \
50 #define _cc_ll_init_extd(_cc_ll, _start, _ratio, _thrsh, _alloc, _ll_)                \
51                                                                                          \
52 STATEMENT_                                                                                   \
53 (                                                                                          \
54     if ((_alloc)) _cont_alloc((_cc_ll));                                                \
55                                                                                          \
56     _##_ll_##_init_core((_cc_ll));                                                       \
57     _##_ll_##_init_info((_cc_ll), (_start), (_ratio), (_thrsh));                       \
58 )                                                                                          \
59                                                                                          \
60                                                                                          \
61 #define _cc_ll_init_core(_cc_ll, _ll_)                                                    \
62                                                                                          \
63 VOID_EXPR_                                                                                   \
64 (                                                                                          \
65     _##_ll_##_init_seed((_cc_ll)),                                                       \
66                                                                                          \
67     (_cc_ll)->avsp = (_cc_ll)->pnode = NULL,                                             \
68     (_cc_ll)->pool = (_cc_ll)->pblock = NULL,                                           \
69                                                                                          \
70     (_cc_ll)->itarr = NULL,                                                               \
71     (_cc_ll)->_iter = NULL,                                                             \
72     (_cc_ll)->_it   = NULL,                                                             \
73     (_cc_ll)->_co   = NULL,                                                             \
74     (_cc_ll)->_it_base = (_cc_ll)->_it_limit = 0,                                       \
75     (_cc_ll)->_co_base = (_cc_ll)->_co_limit = 0                                       \
76 )                                                                                          \
77                                                                                          \
78                                                                                          \
79 #define _cc_ll_init_info(_cc_ll, _start, _ratio, _thrsh)                               \
80                                                                                          \
81 VOID_EXPR_                                                                                   \
82 (                                                                                          \
83     (_cc_ll)->start = ((_start) > 0) ? (_start) : 1,                                   \
84     (_cc_ll)->ratio = ((_ratio) > 0) ? (_ratio) : 1,                                   \
85     (_cc_ll)->thrsh = ((_thrsh) > (_cc_ll)->start) ? (_thrsh) : (_cc_ll)->start\
86 )                                                                                          \
87                                                                                          \
88                                                                                          \
89 #define _cc_ll_iter_init(_iter, _cc_ll, _alloc, _ll_)                                   \
90                                                                                          \
91 STATEMENT_                                                                                   \
92 (                                                                                          \
93     if ((_alloc)) _iter_alloc((_iter));                                                  \
94                                                                                          \
95     _ll_##_iter_init((_iter), (_cc_ll));                                                 \

```

```

96 )
97
98
99
100 /* cc_ll destroy */
101
102
103 #define cc_ll_free(_cc_ll) \
104 \
105 STATEMENT_ \
106 ( \
107     _it_free((_cc_ll)); \
108     _co_free((_cc_ll)); \
109 \
110     _iter_free ((_cc_ll)->_iter); \
111     _itarr_free((_cc_ll)); \
112     _block_free((_cc_ll)); \
113     _cont_free ((_cc_ll)); \
114 )
115
116
117
118 /* cc_ll operations */
119
120
121 #define cc_ll_merge_extd(_cc_ll_d, _cc_ll_s, _leq, _ll_, _opt_) \
122 \
123 STATEMENT_ \
124 ( \
125     if (_unlikely(!_ll_##_empty((_cc_ll_s)))) break; \
126 \
127     _it_init((_cc_ll_d), 1, _base_m1, _ll_); \
128     _it_init((_cc_ll_s), 2, _base_m2, _ll_); \
129 \
130     _##_ll_##_merge##_opt##_extd(_it((_cc_ll_d), _base_m1, 0), \
131                                     _it((_cc_ll_s), _base_m2, 0), \
132                                     _it((_cc_ll_s), _base_m2, 1), _leq); \
133     _it_clear((_cc_ll_d), 1); \
134     _it_clear((_cc_ll_s), 2); \
135 )
136
137
138 #define _cc_ll_merge_extd(_iter_l, _iter_m, _iter_r, _leq, _ll_, _opt_) \
139 \
140 STATEMENT_ \
141 ( \
142     _ll_##_iter_tail ((_iter_l)); \
143     _ll_##_iter_begin((_iter_m)); \
144     _ll_##_iter_tail ((_iter_r)); \
145 \

```

```

146     _ll_##_move_range_extd( (_iter_l), (_iter_m), (_iter_r),          \
147                             _ll_##_size((_iter_m)->cont));          \
148                                                                    \
149     _ll_##_iter_begin((_iter_l));                                    \
150     _ll_##_iter_init ((_iter_r), (_iter_l)->cont);                  \
151     _ll_##_iter_tail ((_iter_r));                                    \
152                                                                    \
153     _ll_##_merge_range##_opt_##_extd((_iter_l), (_iter_m), (_iter_r), _leq); \
154 )                                                                    \
155                                                                    \
156 #define cc_ll_merge_range_extd(_iter_l, _iter_m, _iter_r, _leq, _ll_, _opt_) \
157 STATEMENT_                                                            \
158 (                                                                        \
159     _it_init((_iter_l)->cont, 1, _base_m3, _ll_);                    \
160     _ll_##_merge_range##_opt_##_extd((_iter_l), (_iter_m), (_iter_r), \
161                                         _it((_iter_l)->cont, _base_m3, 0), _leq); \
162     _it_clear((_iter_l)->cont, 1);                                    \
163 )                                                                        \
164                                                                    \
165 #define _cc_ll_merge_range_extd(_iter_l, _iter_m, _iter_r, _iter_x, _leq, _ll_)\
166 STATEMENT_                                                            \
167 (                                                                        \
168     if (_unlikely((_iter_l)->cont != (_iter_m)->cont ||              \
169                     (_iter_m)->cont != (_iter_r)->cont)) break;      \
170     _ll_##_iter_copy((_iter_x), (_iter_m));                          \
171     for (register int _neq; ; )                                         \
172     {                                                                    \
173         while ((_neq = ((_iter_l)->curr.node                          \
174                         != (_iter_m)->curr.node)) && _leq((_iter_l), (_iter_m))) \
175             (void)(_ll_##_iter_incr((_iter_l)));                      \
176         if (!(_neq))                                                    \
177         {                                                                \
178             _ll_##_iter_copy((_iter_l), (_iter_r));                  \
179             _ll_##_iter_copy((_iter_m), (_iter_r)); break;           \
180         }                                                                \
181         (void)(_ll_##_iter_incr((_iter_x)));                          \
182         while ((_neq = ((_iter_x)->curr.node                          \
183                         != (_iter_r)->curr.node)) && !_leq((_iter_l), (_iter_x))) \
184             (void)(_ll_##_iter_incr((_iter_x)));                      \
185     }                                                                    \
186 )                                                                        \
187 
```

```

196                                     \
197     _ll_##_move_range((_iter_l), (_iter_m), (_iter_x));           \
198     _ll_##_iter_copy ((_iter_m), (_iter_x));                       \
199                                     \
200     if (!(_neq))                                                    \
201     {                                                                \
202         _ll_##_iter_copy((_iter_l), (_iter_x));                   \
203         _ll_##_iter_copy((_iter_r), (_iter_x)); break;            \
204     }                                                                \
205 }                                                                    \
206 )                                                                    \
207
208
209 #define cc_ll_sort_extd(_cc_ll, _leq, _ll_, _opt_)                 \
210                                     \
211 STATEMENT_                                                           \
212 (                                                                     \
213     if (_unlikely(_ll_##_size((_cc_ll)) <= 1)) break;             \
214                                     \
215     int _buck = (int)(log2(_ll_##_size((_cc_ll)))) + 1;            \
216                                     \
217     _co_init((_cc_ll), 1 + _buck, _base_s1, _ll_);                 \
218     _it_init((_cc_ll), 2, _base_s2, _ll_);                         \
219                                     \
220     _##_ll_##_sort##_opt##_extd(      (_cc_ll),                   \
221                                         _co((_cc_ll), _base_s1, 0), \
222                                         &(_co((_cc_ll), _base_s1, 1)), \
223                                         _it((_cc_ll), _base_s2, 0), \
224                                         _it((_cc_ll), _base_s2, 1), _leq); \
225                                     \
226     _co_clear((_cc_ll), _buck + 1);                                 \
227     _it_clear((_cc_ll), 2);                                         \
228 )                                                                    \
229
230
231 #define _cc_ll_sort_extd(_cc_ll, _carry, _pcont,                   \
232                         _iter_a, _iter_b, _leq, _ll_, _opt_)       \
233 STATEMENT_                                                           \
234 (                                                                     \
235     int _fill = 0, _curr;                                           \
236                                     \
237     do                                                                \
238     {                                                                \
239         _ll_##_iter_init ((_iter_a), (_carry));                   \
240         _ll_##_iter_init ((_iter_b), (_cc_ll));                   \
241         _ll_##_move_begin((_iter_a), (_iter_b));                   \
242                                     \
243         for (_curr = 0; _curr != _fill &&                           \
244             !(_ll_##_empty((_pcont)[_curr])); _curr++)             \
245         {                                                            \
246             _ll_##_merge##_opt##_extd((_pcont)[_curr], (_carry), _leq);

```

```

246         _ll_##_swap((_pcont)[_curr], (_carry)); \
247     } \
248     _ll_##_swap((_pcont)[_curr], (_carry)); \
249 \
250     if (_unlikely(_curr == _fill)) _fill++; \
251 } \
252 while (!(_ll_##_empty((_cc_ll)))); \
253 \
254 for (_curr = 0; _curr < _fill; _curr++) \
255     _ll_##_merge##_opt##_extd((_cc_ll), (_pcont)[_curr], _leq); \
256 ) \
257 \
258 \
259 /* cc_ll iterators */
260
261
262 #define cc_ll_iter_distance(_iter_a, _iter_b, _pdist, _ll_ /* != ccsll */ ) \
263 \
264 STATEMENT_ \
265 ( \
266     _it_init((_iter_a)->cont, 1, _base_d1, _ll_); \
267 \
268     _ll_##_iter_copy(_it((_iter_a)->cont, _base_d1, 0), (_iter_a)); \
269 \
270 STATEMENT_ \
271 ( \
272     (*(_pdist)) = 0; \
273     if ((_iter_a)->cont != (_iter_b)->cont) break; \
274 \
275     while ((_iter_a)->curr.node != (_iter_b)->curr.node && ++(*(_pdist))) \
276         if (!(_ll_##_iter_incr((_iter_a)))) break; \
277 \
278     if ((_iter_a)->curr.node == (_iter_b)->curr.node) break; \
279     else (*(_pdist)) = 0; \
280 \
281     _ll_##_iter_copy((_iter_a), _it((_iter_a)->cont, _base_d1, 0)); \
282 \
283     while ((_iter_a)->curr.node != (_iter_b)->curr.node && --(*(_pdist))) \
284         if (!(_ll_##_iter_decr((_iter_a)))) break; \
285 \
286     if ((_iter_a)->curr.node == (_iter_b)->curr.node) break; \
287     else (*(_pdist)) = 0; \
288 ) \
289 \
290 _ll_##_iter_copy((_iter_a), _it((_iter_a)->cont, _base_d1, 0)); \
291 \
292 _it_clear((_iter_a)->cont, 1); \
293 ) \
294 \
295

```

```

296
297 #define cc_ll_iter_distance_positive(_iter_a, _iter_b, _pdist, _ll_) \
298 \
299 STATEMENT_ \
300 ( \
301     _it_init((_iter_a)->cont, 1, _base_d1, _ll_); \
302 \
303     _ll_##_iter_copy(_it((_iter_a)->cont, _base_d1, 0), (_iter_a)); \
304 \
305     STATEMENT_ \
306     ( \
307         (*(_pdist)) = 0; \
308         if ((_iter_a)->cont != (_iter_b)->cont) break; \
309 \
310         while ((_iter_a)->curr.node != (_iter_b)->curr.node && ++(*(_pdist))) \
311             if (!(_ll_##_iter_incr((_iter_a)))) break; \
312 \
313         if ((_iter_a)->curr.node == (_iter_b)->curr.node) break; \
314         else (*(_pdist)) = 0; \
315     ); \
316 \
317     _ll_##_iter_copy((_iter_a), _it((_iter_a)->cont, _base_d1, 0)); \
318 \
319     _it_clear((_iter_a)->cont, 1); \
320 ) \
321 \
322 \
323 \
324 /* cc_ll traversor */ \
325 \
326 \
327 #define CC_LL_INCR(_iter, _ll_) \
328 \
329     for (_ll_##_iter_head((_iter)); _ll_##_iter_incr((_iter)); ) \
330 \
331 \
332 #define CC_LL_INCR_EXTD(_pval, _cc_ll, _ll_, ...) \
333 \
334     for (__typeof__((_cc_ll)->pnode->val) *_pval, \
335         *_init = (_ll_##_iter_head((_cc_ll)->_iter), NULL); \
336         (_ll_##_iter_incr((_cc_ll)->_iter)) && \
337         ((_pval) = &(LREF((_cc_ll)->_iter)), 1); (__VA_ARGS__), (void)_init) \
338 \
339 \
340 #define CC_LL_DECR(_iter, _ll_) \
341 \
342     for (_ll_##_iter_tail((_iter)); _ll_##_iter_decr((_iter)); ) \
343 \
344 \
345 #define CC_LL_DECR_EXTD(_pval, _cc_ll, _ll_, ...) \

```



```

346                                                                 \
347     for (__typeof__((_cc_ll)->pnode->val) *_pval,                \
348         *_init = (_ll_##_iter_tail((_cc_ll)->_iter), NULL);      \
349         (_ll_##_iter_decr((_cc_ll)->_iter)) &&                  \
350         ((_pval) = &(LREF((_cc_ll)->_iter)), 1); (__VA_ARGS__), (void)_init)
351
352
353
354 #endif

```

1.1.2 list/ccxll.h

```
1  #ifndef OPENG3_LIST_CCXLL_H
2  #define OPENG3_LIST_CCXLL_H
3
4  #include "base.h"
5  #include "../base/pool.h"
6  #include "../base/misc.h"
7  #include "../base/snym.h"
8
9  #include <math.h>
10 #include <stddef.h>
11 #include <stdint.h>
12 #include <string.h>
13
14
15 /* ccxll create */
16
17
18 #define ccxll(elem_t) \
19 \
20     ccxll_extd(elem_t, 1, NORMAL)
21
22 #define ccxll_pckd(elem_t) \
23 \
24     ccxll_extd(elem_t, 1, PACKED)
25
26 #define ccxll_extd(elem_t, _n_iter, _ALIGN_) \
27 \
28     typedef ccxll_struct_extd(elem_t, _n_iter, _ALIGN_) *CCXLL; CCXLL
29
30 #define ccxll_type(elem_t) \
31 \
32     typedef ccxll_struct_extd(elem_t, 1, NORMAL) *
33
34
35 #define link_t void*
36
37 #define ccxll_struct(elem_t) \
38 \
39     ccxll_struct_extd(elem_t, 1, NORMAL)
40
41 #define ccxll_struct_pckd(elem_t) \
42 \
43     ccxll_struct_extd(elem_t, 1, PACKED)
44
45 #define ccxll_struct_extd(elem_t, _n_iter, _ALIGN_) \
46 \
47     struct CCXLL_CONT \
48     {
```

```

49     int size, last, vcnt;                /* size and node record */ \
50     int start, ratio, thrsh;            /* block increment info */ \
51                                         \
52     struct CCXLL_NODE                   \
53     {   link_t lnk[1];                  \
54         elem_t val;                     /* val with an xor link */ \
55     }   *avsp, *pnode, swap;            /* available space list */ \
56                                         \
57     union CCXLL_HDTL                   \
58     {   link_t lnk[1];                  \
59         struct CCXLL_NODE *stnl;        /* points to same addr. */ \
60     }   head, tail;                     /* two pseudo sentinels */ \
61                                         \
62     struct CCXLL_BLK                     \
63     {   struct CCXLL_BLK *bprv, *bnxt;  /* next and prev blocks */ \
64         PRAGMA_##_ALIGN_##_BGN          /* packed pragma starts */ \
65         int ncnt;                       /* the item of the node */ \
66         struct CCXLL_NODE nodes[];      /* node structure array */ \
67         PRAGMA_##_ALIGN_##_END          /* the pragma ends here */ \
68     }   *pool, *pblock;                 /* points to 1-st block */ \
69                                         \
70     struct CCXLL_ITER                   \
71     {   union CCXLL_PTRS                \
72         {   link_t lnk[1];              \
73             struct CCXLL_NODE *node;    \
74         }   prev, curr, next;           /* points to p/v/n node */ \
75         struct CCXLL_CONT *cont;        /* points to ccxll body */ \
76     }   (*itarr)[_n_iter], *_iter, **_it; \
77                                         \
78     struct CCXLL_CONT **_co;            /* internal use _it _co */ \
79                                         \
80     int _it_base, _it_limit;             \
81     int _co_base, _co_limit;             \
82 }
83
84
85
86 /* ccxll initialize */
87
88
89 #define ccxll_init(_ccxll)              \
90                                         \
91     ccxll_init_extd(_ccxll,    16,      2,  65536)
92
93 #define ccxll_init_extd(_ccxll, _start, _ratio, _thrsh) \
94                                         \
95     cc_ll_init_extd(_ccxll, _start, _ratio, _thrsh, ccxll)
96
97
98 #define _ccxll_init(_ccxll_dst, _ccxll_src, _alloc) \

```

```

99                                     \
100         _ccxll_init_extd(_ccxll_dst, -1,      -1,      -1, _alloc)
101
102 #define _ccxll_init_extd(_ccxll, _start, _ratio, _thrsh, _alloc)           \
103                                     \
104         _cc_ll_init_extd(_ccxll, _start, _ratio, _thrsh, _alloc, ccxll)
105
106
107 #define _ccxll_init_core(_ccxll)                                           \
108                                     \
109         _cc_ll_init_core(_ccxll, ccxll)
110
111
112 #define _ccxll_init_seed(_ccxll)                                           \
113                                     \
114 VOID_EXPR_                                                                  \
115 (                                                                            \
116     (_ccxll)->size = 0,                                                    \
117     (_ccxll)->last = (_ccxll)->vcnt = 0,                                  \
118                                     \
119     (_ccxll)->head.XOR = &((_ccxll)->tail),                               \
120     (_ccxll)->tail.XOR = &((_ccxll)->head)                                \
121 )
122
123
124 #define _ccxll_init_info(_ccxll, _start, _ratio, _thrsh)                 \
125                                     \
126         _cc_ll_init_info(_ccxll, _start, _ratio, _thrsh)
127
128
129 #define ccxll_iter_init(_iter, _ccxll)                                     \
130                                     \
131 VOID_EXPR_                                                                  \
132 (                                                                            \
133     (_iter)->prev.XOR = NULL,                                              \
134     (_iter)->curr.XOR = NULL,                                              \
135     (_iter)->next.XOR = NULL,                                              \
136     (_iter)->cont = (_ccxll)                                               \
137 )
138
139
140 #define _ccxll_iter_init(_iter, _ccxll, _alloc)                          \
141                                     \
142         _cc_ll_iter_init(_iter, _ccxll, _alloc, ccxll)
143
144
145
146 /* ccxll destroy */
147
148

```

```

149 #define ccxll_free(_ccxll)  cc_ll_free(_ccxll)
150
151
152
153 /* ccxll access */
154
155
156 #define ccxll_front(_ccxll)  ((_ccxll)->head.stnl->val)
157
158 #define ccxll_back(_ccxll)   ((_ccxll)->tail.stnl->val)
159
160
161
162 /* ccxll capacity */
163
164
165 #define ccxll_size(_ccxll)   ((_ccxll)->size)
166
167 #define ccxll_empty(_ccxll)  ((_ccxll)->head.XOR == &((_ccxll)->tail) && \
168                               (_ccxll)->tail.XOR == &((_ccxll)->head))
169
170
171
172 /* ccxll modifiers */
173
174
175 #define ccxll_push_front(_ccxll, _val)  _ccxll_push(_ccxll, _val, head)
176
177 #define ccxll_push_back(_ccxll, _val)   _ccxll_push(_ccxll, _val, tail)
178
179 #define _ccxll_push(_ccxll, _val, _hdtl_) \
180 \
181 STATEMENT_ \
182 ( \
183     _ccxll_push_alloc((_ccxll), _hdtl_); \
184 \
185     (_ccxll)->pnode->val = (_val); \
186 )
187
188
189 #define ccxll_push_front_alloc(_ccxll) _ccxll_push_alloc(_ccxll, head)
190
191 #define ccxll_push_back_alloc(_ccxll)  _ccxll_push_alloc(_ccxll, tail)
192
193 #define _ccxll_push_alloc(_ccxll, _hdtl_) \
194 \
195 STATEMENT_ \
196 ( \
197     _node_alloc((_ccxll)->pnode, (_ccxll)); \
198 \

```

```

199     (_ccxll)->pnode->XOR          = XOR2(&((_ccxll)->_hdtl_.XOR),
200                                     (_ccxll)->_hdtl_.XOR);
201
202     (_ccxll)->_hdtl_.stnl->XOR = XOR3(&((_ccxll)->_hdtl_.XOR),
203                                     (_ccxll)->_hdtl_.stnl->XOR,
204                                     &((_ccxll)->pnode->XOR));
205
206     (_ccxll)->_hdtl_.XOR = &((_ccxll)->pnode->XOR);
207
208     (_ccxll)->size++;
209 )
210
211
212 #define ccxll_pop_front(_ccxll)  _ccxll_pop(_ccxll, head)
213
214 #define ccxll_pop_back(_ccxll)   _ccxll_pop(_ccxll, tail)
215
216 #define _ccxll_pop(_ccxll, _hdtl_)
217
218 STATEMENT_
219 (
220     if (ccxll_empty((_ccxll))) break;
221
222     (_ccxll)->pnode = (_ccxll)->_hdtl_.stnl;
223
224     (_ccxll)->_hdtl_.XOR          = XOR2(&((_ccxll)->_hdtl_.XOR),
225                                     (_ccxll)->_hdtl_.stnl->XOR);
226
227     (_ccxll)->_hdtl_.stnl->XOR = XOR3(&((_ccxll)->_hdtl_.XOR),
228                                     (_ccxll)->_hdtl_.stnl->XOR,
229                                     &((_ccxll)->pnode->XOR));
230
231     _node_clear((_ccxll)->pnode, (_ccxll));
232
233     (_ccxll)->size--;
234 )
235
236
237 #define ccxll_insert(_iter, _val)
238
239 STATEMENT_
240 (
241     if (ccxll_iter_at_head((_iter))) break;
242
243     _node_alloc((_iter)->cont->pnode, (_iter)->cont);
244
245     (_iter)->cont->pnode->val = (_val);
246
247     (_iter)->next.node = (_iter)->curr.node;
248     (_iter)->curr.node = (_iter)->cont->pnode;

```

```

249                                     \
250     (_iter)->curr.node->XOR = XOR2((_iter)->prev.XOR, (_iter)->next.XOR); \
251                                     \
252     (_iter)->prev.node->XOR = XOR3((_iter)->prev.node->XOR, \
253                                     (_iter)->next.XOR, \
254                                     &((_iter)->cont->pnode->XOR)); \
255     (_iter)->next.node->XOR = XOR3((_iter)->next.node->XOR, \
256                                     (_iter)->prev.XOR, \
257                                     &((_iter)->cont->pnode->XOR)); \
258     (_iter)->cont->size++; \
259 ) \
260 \
261 \
262 #define ccxll_erase(_iter) \
263 \
264 STATEMENT_ \
265 ( \
266     if (ccxll_iter_at_head((_iter)) || ccxll_iter_at_tail((_iter))) break; \
267 \
268     (_iter)->prev.node->XOR = XOR3((_iter)->prev.node->XOR, \
269                                     (_iter)->next.XOR, (_iter)->curr.XOR); \
270     (_iter)->next.node->XOR = XOR3((_iter)->next.node->XOR, \
271                                     (_iter)->prev.XOR, (_iter)->curr.XOR); \
272 \
273     _node_clear((_iter)->curr.node, (_iter)->cont); \
274 \
275     (_iter)->curr.XOR = (_iter)->next.XOR; \
276     (_iter)->next.XOR = XOR2((_iter)->curr.node->XOR, (_iter)->prev.XOR); \
277 \
278     (_iter)->cont->size--; \
279 ) \
280 \
281 \
282 #define ccxll_swap(_ccxll_a, _ccxll_b) \
283 \
284 STATEMENT_ \
285 ( \
286     XOR2_SWAP((_ccxll_a), (_ccxll_b)); \
287 ) \
288 \
289 \
290 #define ccxll_resize(_ccxll, _items, _val) \
291 \
292 STATEMENT_ \
293 ( \
294     int _size = ccxll_size((_ccxll)) - (_items); \
295 \
296     if (_size > 0) while(_size--) ccxll_pop_back((_ccxll)); \
297     else if (_size < 0) while(_size++) ccxll_push_back((_ccxll), (_val)); \
298 )

```

```

299
300
301 #define ccxll_clear(_ccxll) \
302 \
303 STATEMENT_ \
304 ( \
305     while (!(ccxll_empty((_ccxll))))    ccxll_pop_back ((_ccxll)); \
306 )
307
308
309
310 /* ccxll operations */
311
312
313 #define ccxll_move(_iter_p, _iter_i) \
314 \
315 STATEMENT_ \
316 ( \
317     if (_unlikely((_iter_i)->curr.node == (_iter_p)->prev.node)) break; \
318 \
319     if (_unlikely(ccxll_iter_at_head((_iter_p)) || \
320                 ccxll_iter_at_head((_iter_i)) || \
321                 ccxll_iter_at_tail((_iter_i)))) break; \
322 \
323     _ccxll_move((_iter_p), (_iter_i)); \
324 \
325     (_iter_i)->cont->size--; \
326     (_iter_i)->cont = (_iter_p)->cont; \
327     (_iter_i)->cont->size++; \
328 )
329
330 #define _ccxll_move(_iter_p, _iter_i) \
331 \
332 STATEMENT_ \
333 ( \
334     (_iter_i)->prev.node->XOR = XOR3((_iter_i)->prev.node->XOR, \
335                                     (_iter_i)->next.XOR, (_iter_i)->curr.XOR); \
336     (_iter_i)->next.node->XOR = XOR3((_iter_i)->next.node->XOR, \
337                                     (_iter_i)->prev.XOR, (_iter_i)->curr.XOR); \
338 \
339     (_iter_i)->prev.XOR = (_iter_p)->prev.XOR; \
340     (_iter_i)->next.XOR = (_iter_p)->curr.XOR; \
341 \
342     (_iter_i)->curr.node->XOR = XOR2((_iter_i)->prev.XOR, (_iter_i)->next.XOR); \
343 \
344     (_iter_i)->prev.node->XOR = XOR3((_iter_i)->prev.node->XOR, \
345                                     (_iter_i)->next.XOR, (_iter_i)->curr.XOR); \
346     (_iter_i)->next.node->XOR = XOR3((_iter_i)->next.node->XOR, \
347                                     (_iter_i)->prev.XOR, (_iter_i)->curr.XOR); \
348 \

```



```

349     (_iter_p)->prev.XOR =      (_iter_i)->curr.XOR;          \
350     (_iter_p)->next.XOR = XOR2((_iter_p)->prev.XOR, (_iter_p)->curr.node->XOR);\
351 )
352
353
354 #define ccxll_move_begin(_iter_a, _iter_b)                    \
355                                                                \
356 STATEMENT_                                                    \
357 (                                                            \
358     ccxll_iter_begin((_iter_a));                             \
359     ccxll_iter_begin((_iter_b));                             \
360                                                                \
361     ccxll_move((_iter_a), (_iter_b));                         \
362 )
363
364
365 #define ccxll_move_range(_iter_p, _iter_l, _iter_r)          \
366                                                                \
367     ccxll_move_range_extd(_iter_p, _iter_l, _iter_r,    -1)
368
369 #define ccxll_move_range_extd(_iter_p, _iter_l, _iter_r, _dist) /* [l, r) */ \
370                                                                \
371 STATEMENT_                                                    \
372 (                                                            \
373     if (_unlikely(ccxll_iter_at_head ((_iter_p)) ||         \
374                 ccxll_iter_at_head ((_iter_l)))) break;    \
375                                                                \
376     if (_unlikely((_iter_l)->curr.node == (_iter_r)->curr.node)) break; \
377                                                                \
378     if (_unlikely((_iter_l)->cont != (_iter_r)->cont)) break; \
379                                                                \
380     if (_unlikely((_iter_p)->cont != (_iter_l)->cont))      \
381     {                                                         \
382         int _dist_m = (_dist);                               \
383                                                                \
384         if (_dist_m < 0)                                       \
385             ccxll_iter_distance((_iter_l), (_iter_r), &_dist_m); \
386                                                                \
387         if (_dist_m <= 0) break;                               \
388                                                                \
389         (_iter_p)->cont->size += _dist_m;                     \
390         (_iter_l)->cont->size -= _dist_m;                     \
391         (_iter_l)->cont = (_iter_p)->cont;                   \
392     }                                                         \
393
394 link_t _p_c = (_iter_p)->curr.XOR;
395 link_t _l_c = (_iter_l)->curr.XOR;
396 link_t _r_c = (_iter_r)->curr.XOR;
397 link_t _p_p = (_iter_p)->prev.XOR;
398 link_t _l_p = (_iter_l)->prev.XOR;

```

```

399     link_t _r_p = (_iter_r)->prev.XOR; \
400 \
401     (_iter_p)->prev.node->XOR = XOR3((_iter_p)->prev.node->XOR, _p_c, _l_c); \
402     (_iter_l)->prev.node->XOR = XOR3((_iter_l)->prev.node->XOR, _l_c, _r_c); \
403     (_iter_r)->prev.node->XOR = XOR3((_iter_r)->prev.node->XOR, _r_c, _p_c); \
404 \
405     (_iter_p)->curr.node->XOR = XOR3((_iter_p)->curr.node->XOR, _p_p, _r_p); \
406     (_iter_l)->curr.node->XOR = XOR3((_iter_l)->curr.node->XOR, _l_p, _p_p); \
407     (_iter_r)->curr.node->XOR = XOR3((_iter_r)->curr.node->XOR, _r_p, _l_p); \
408 \
409     if (_unlikely((_iter_p)->next.XOR == _l_c)) (_iter_p)->next.XOR = _r_c; \
410     if ((_iter_l)->next.XOR == _r_c) (_iter_l)->next.XOR = _p_c; \
411     if (_unlikely((_iter_r)->next.XOR == _p_c)) (_iter_r)->next.XOR = _l_c; \
412 \
413     (_iter_p)->prev.XOR = XOR2((_iter_p)->curr.node->XOR, (_iter_p)->next.XOR);\
414     (_iter_l)->prev.XOR = XOR2((_iter_l)->curr.node->XOR, (_iter_l)->next.XOR);\
415     (_iter_r)->prev.XOR = XOR2((_iter_r)->curr.node->XOR, (_iter_r)->next.XOR);\
416 )
417
418
419 #define ccxll_merge(_ccxll_d, _ccxll_s) \
420 \
421     ccxll_merge_extd(_ccxll_d, _ccxll_s, XLEQ)
422
423 #define ccxll_merge_extd(_ccxll_d, _ccxll_s, _leq) \
424 \
425     cc_ll_merge_extd(_ccxll_d, _ccxll_s, _leq, ccxll, )
426
427 #define _ccxll_merge_extd(_iter_l, _iter_m, _iter_r, _leq) \
428 \
429     _cc_ll_merge_extd(_iter_l, _iter_m, _iter_r, _leq, ccxll, )
430
431
432 #define ccxll_merge_range(_iter_l, _iter_m, _iter_r) \
433 \
434     ccxll_merge_range_extd(_iter_l, _iter_m, _iter_r, XLEQ)
435
436 #define ccxll_merge_range_extd(_iter_l, _iter_m, _iter_r, _leq) \
437 \
438     cc_ll_merge_range_extd(_iter_l, _iter_m, _iter_r, _leq, ccxll, )
439
440 #define _ccxll_merge_range_extd(_iter_l, _iter_m, _iter_r, _iter_x, _leq) \
441 \
442     _cc_ll_merge_range_extd(_iter_l, _iter_m, _iter_r, _iter_x, _leq, ccxll)
443
444
445 #define ccxll_sort(_ccxll) \
446 \
447     ccxll_sort_extd(_ccxll, XLEQ)
448

```

```

449 #define ccxll_sort_extd(_ccxll, _leq) \
450 \
451     cc_ll_sort_extd(_ccxll, _leq, ccxll, )
452
453 #define _ccxll_sort_extd(_ccxll, _carry, _pbuck, _iter_a, _iter_b, _leq) \
454 \
455     _cc_ll_sort_extd(_ccxll, _carry, _pbuck, _iter_a, _iter_b, _leq, ccxll,)
456
457
458 #define ccxll_reverse_range(_iter_l, _iter_r) \
459 \
460     ccxll_reverse_range_extd(_iter_l, _iter_r, 0)
461
462 #define ccxll_reverse_range_extd(_iter_l, _iter_r, _flag_swap_iters) \
463 \
464 STATEMENT_ \
465 ( \
466     if (_unlikely((_iter_l)->cont != (_iter_r)->cont)) break; \
467 \
468     if (_unlikely((_iter_l)->curr.XOR == (_iter_r)->curr.XOR)) break; \
469 \
470     link_t _x_in = XOR2((_iter_l)->curr.XOR, (_iter_r)->curr.XOR); \
471     link_t _x_ex = XOR2((_iter_l)->prev.XOR, (_iter_r)->next.XOR); \
472 \
473     (_iter_l)->prev.node->XOR = XOR2((_iter_l)->prev.node->XOR, _x_in); \
474     (_iter_r)->next.node->XOR = XOR2((_iter_r)->next.node->XOR, _x_in); \
475 \
476     (_iter_l)->curr.node->XOR = XOR2((_iter_l)->curr.node->XOR, _x_ex); \
477     (_iter_r)->curr.node->XOR = XOR2((_iter_r)->curr.node->XOR, _x_ex); \
478 \
479     switch ((_flag_swap_iters)) \
480     { \
481         case 0: \
482             XOR2_SWAP((_iter_l)->prev.XOR, (_iter_l)->next.XOR); \
483             XOR2_SWAP((_iter_r)->prev.XOR, (_iter_r)->next.XOR); \
484             (_iter_l)->next.XOR = XOR2((_iter_l)->next.XOR, _x_ex); \
485             (_iter_r)->prev.XOR = XOR2((_iter_r)->prev.XOR, _x_ex); \
486             break; \
487 \
488         case 1: default: \
489             XOR2_SWAP((_iter_l)->curr.XOR, (_iter_r)->curr.XOR); \
490             XOR2_SWAP((_iter_l)->next.XOR, (_iter_r)->prev.XOR); \
491     } \
492 )
493
494
495
496 /* ccxll comparator */
497
498

```

```

499 #define ccxll_comp_leq(_iter_a, _iter_b)      (XREF((_iter_a)) <=      \
500                                                XREF((_iter_b)))
501
502 #define ccxll_comp_leq_prev(_iter_a, _iter_b) (XREF_PREV((_iter_a)) <=  \
503                                                XREF_PREV((_iter_b)))
504
505 #define ccxll_comp_leq_next(_iter_a, _iter_b) (XREF_NEXT((_iter_a)) <=  \
506                                                XREF_NEXT((_iter_b)))
507
508 #define ccxll_comp_geq(_iter_a, _iter_b)      (XREF((_iter_a)) >=      \
509                                                XREF((_iter_b)))
510
511 #define ccxll_comp_geq_prev(_iter_a, _iter_b) (XREF_PREV((_iter_a)) >=  \
512                                                XREF_PREV((_iter_b)))
513
514 #define ccxll_comp_geq_next(_iter_a, _iter_b) (XREF_NEXT((_iter_a)) >=  \
515                                                XREF_NEXT((_iter_b)))
516
517
518 /* ccxll iterators */
519
520
521 #define ccxll_iter_copy(_iter_dst, _iter_src)      \
522 VOID_EXPR_                                         \
523 (                                                  \
524     *(_iter_dst) = *(_iter_src)                  \
525 )
526
527
528
529
530 #define ccxll_iter_head(_iter)                    \
531 VOID_EXPR_                                         \
532 (                                                  \
533     (_iter)->prev.XOR = NULL,                      \
534     (_iter)->curr.XOR = &((_iter)->cont->head.XOR), \
535     (_iter)->next.XOR = &((_iter)->cont->head.stnl->XOR) \
536 )
537
538
539
540 #define ccxll_iter_tail(_iter)                    \
541 VOID_EXPR_                                         \
542 (                                                  \
543     (_iter)->next.XOR = NULL,                      \
544     (_iter)->curr.XOR = &((_iter)->cont->tail.XOR), \
545     (_iter)->prev.XOR = &((_iter)->cont->tail.stnl->XOR) \
546 )
547
548

```

```

549
550 #define ccxll_iter_begin(_iter) \
551 ( \
552     (_iter)->prev.XOR =      &((_iter)->cont->head.XOR), \
553     (_iter)->curr.XOR =      &((_iter)->cont->head.stn1->XOR), \
554     (_iter)->next.XOR = XOR2(&((_iter)->cont->head.XOR), \
555                             (_iter)->cont->head.stn1->XOR) \
556 )
557
558
559 #define ccxll_iter_end(_iter) \
560 ( \
561     (_iter)->next.XOR =      &((_iter)->cont->tail.XOR), \
562     (_iter)->curr.XOR =      &((_iter)->cont->tail.stn1->XOR), \
563     (_iter)->prev.XOR = XOR2(&((_iter)->cont->tail.XOR), \
564                             (_iter)->cont->tail.stn1->XOR) \
565 )
566
567
568 #define ccxll_iter_at_head(_iter) ( (_iter)->curr.XOR == \
569                                     &((_iter)->cont->head.XOR))
570
571 #define ccxll_iter_at_tail(_iter) ( (_iter)->curr.XOR == \
572                                     &((_iter)->cont->tail.XOR))
573
574 #define ccxll_iter_at_begin(_iter) ( (_iter)->curr.XOR == \
575                                     (_iter)->cont->head.XOR )
576
577 #define ccxll_iter_at_end(_iter) ( (_iter)->curr.XOR == \
578                                     (_iter)->cont->tail.XOR )
579
580
581 #define ccxll_iter_incr(_iter) \
582 ( \
583     ccxll_iter_at_tail((_iter)) ? (NULL) : \
584     ( \
585         (_iter)->prev.XOR = (_iter)->curr.XOR, \
586         (_iter)->curr.XOR = (_iter)->next.XOR, \
587         \
588         _prefetch((_iter)->next.XOR = XOR2( (_iter)->prev.XOR, \
589                                             (_iter)->curr.node->XOR)), (_iter)->next.XOR \
590     ) \
591 )
592
593
594 #define ccxll_iter_decr(_iter) \
595 ( \
596     ccxll_iter_at_head((_iter)) ? (NULL) : \
597     ( \
598         (_iter)->next.XOR = (_iter)->curr.XOR, \

```

```

599         (_iter)->curr.XOR = (_iter)->prev.XOR, \
600 \
601         _prefetch((_iter)->prev.XOR = XOR2( (_iter)->next.XOR, \
602             (_iter)->curr.node->XOR)), (_iter)->prev.XOR \
603     ) \
604 )
605
606
607 #define ccxll_iter_advance(_iter, _diff) \
608 \
609 STATEMENT_ \
610 ( \
611     int _len = (_diff); \
612 \
613     if (_len > 0) while (ccxll_iter_incr((_iter)) && --_len); \
614     else if (_len < 0) while (ccxll_iter_decr((_iter)) && ++_len); \
615 )
616
617
618 #define ccxll_iter_distance(_iter_a, _iter_b, _pdist) \
619 \
620     cc_ll_iter_distance(_iter_a, _iter_b, _pdist, ccxll)
621
622
623 #ifndef CC_STRICT
624
625 #define ccxll_iter_swap(_iter_a, _iter_b) \
626 \
627 VOID_EXPR_ \
628 ( \
629     XOR2_SWAP((_iter_a)->prev.node, (_iter_b)->prev.node), \
630     XOR2_SWAP((_iter_a)->curr.node, (_iter_b)->curr.node), \
631     XOR2_SWAP((_iter_a)->next.node, (_iter_b)->next.node) \
632 )
633
634 #else
635
636 #define ccxll_iter_swap(_iter_a, _iter_b) \
637 \
638 VOID_EXPR_ \
639 ( \
640     XOR2_SWAP((_iter_a), (_iter_b)) \
641 )
642
643 #endif // CC_STRICT
644
645
646
647 /* ccxll traversor */
648

```

```

649
650 #define CCXLL_INCR(_iter)  CC_LL_INCR(_iter, ccxll)
651
652 #ifndef CC_STRICT
653
654 #define CCXLL_INCR_AUTO(_pval, _ccxll) \
655 \
656         CCXLL_INCR_EXTD(_pval, _ccxll, (void)0)
657
658 #define CCXLL_INCR_EXTD(_pval, _ccxll, ...) \
659 \
660         CC_LL_INCR_EXTD(_pval, _ccxll, ccxll, __VA_ARGS__)
661
662 #endif // CC_STRICT
663
664
665 #define CCXLL_DECR(_iter)  CC_LL_DECR(_iter, ccxll)
666
667 #ifndef CC_STRICT
668
669 #define CCXLL_DECR_AUTO(_pval, _ccxll) \
670 \
671         CCXLL_DECR_EXTD(_pval, _ccxll, (void)0)
672
673 #define CCXLL_DECR_EXTD(_pval, _ccxll, ...) \
674 \
675         CC_LL_DECR_EXTD(_pval, _ccxll, ccxll, __VA_ARGS__)
676
677 #endif // CC_STRICT
678
679
680
681 #endif

```

1.1.3 list/ccdll.h

```
1  #ifndef OPENG3_LIST_CCDLL_H
2  #define OPENG3_LIST_CCDLL_H
3
4  #include "base.h"
5  #include "../base/pool.h"
6  #include "../base/misc.h"
7  #include "../base/snym.h"
8
9  #include <math.h>
10 #include <stddef.h>
11 #include <stdint.h>
12 #include <string.h>
13
14
15 /* ccdll create */
16
17
18 #define ccdll(elem_t) \
19 \
20     ccdll_extd(elem_t, 1, NORMAL)
21
22 #define ccdll_pckd(elem_t) \
23 \
24     ccdll_extd(elem_t, 1, PACKED)
25
26 #define ccdll_extd(elem_t, _n_iter, _ALIGN_) \
27 \
28     typedef ccdll_struct_extd(elem_t, _n_iter, _ALIGN_) *CCDLL; CCDLL
29
30 #define ccdll_type(elem_t) \
31 \
32     typedef ccdll_struct_extd(elem_t, 1, NORMAL) *
33
34
35 #define ccdll_struct(elem_t) \
36 \
37     ccdll_struct_extd(elem_t, 1, NORMAL)
38
39 #define ccdll_struct_pckd(elem_t) \
40 \
41     ccdll_struct_extd(elem_t, 1, PACKED)
42
43 #define ccdll_struct_extd(elem_t, _n_iter, _ALIGN_) \
44 \
45     struct CCDLL_CONT \
46     { \
47         int size, last, vcnt; /* size and node record */ \
48         int start, ratio, thrsh; /* block increment info */ \
49     }
```



```

49                                     \
50 struct CCDLL_NODE                  \
51 { struct CCDLL_NODE *lnk[2];      \
52     elem_t val;                    /* val with prv and nxt */ \
53 } *avsp, *pnode, head, tail, swap; /* available space list */ \
54                                     \
55 struct CCDLL_BLK                    \
56 { struct CCDLL_BLK *bprv, *bnxt;  /* points to prev block */ \
57     int ncnt;                      /* the item of the node */ \
58     PRAGMA_##_ALIGN_##_BGN         /* packed pragma starts */ \
59     struct CCDLL_NODE nodes[];     /* node structure array */ \
60     PRAGMA_##_ALIGN_##_END         /* the pragma ends here */ \
61 } *pool, *pblock;                  /* points to 1-st block */ \
62                                     \
63 struct CCDLL_ITER                  \
64 { struct CCDLL_PTRS                \
65     { struct CCDLL_NODE *node;     \
66     } curr;                        /* points to curr. node */ \
67     struct CCDLL_CONT *cont;       /* points to ccdll body */ \
68 } (*itarr)[_n_iter], *_iter, **_it; \
69                                     \
70 struct CCDLL_CONT **_co;           /* internal use _it _co */ \
71                                     \
72 int _it_base, _it_limit;            \
73 int _co_base, _co_limit;           \
74 }
75
76
77
78 /* ccdll initialize */
79
80
81 #define ccdll_init(_ccdll)          \
82                                     \
83     ccdll_init_extd(_ccdll,      16,      2,  65536)
84
85 #define ccdll_init_extd(_ccdll, _start, _ratio, _thrsh) \
86                                     \
87     cc_ll_init_extd(_ccdll, _start, _ratio, _thrsh, ccdll)
88
89
90 #define _ccdll_init(_ccdll_dst, _ccdll_src, _alloc) \
91                                     \
92     _ccdll_init_extd(_ccdll_dst, -1,      -1,      -1, _alloc)
93
94 #define _ccdll_init_extd(_ccdll, _start, _ratio, _thrsh, _alloc) \
95                                     \
96     _cc_ll_init_extd(_ccdll, _start, _ratio, _thrsh, _alloc, ccdll)
97
98

```

```

99  #define _ccd11_init_core(_ccd11)                                \
100                                                                    \
101      _cc_11_init_core(_ccd11, ccd11)
102
103
104  #define _ccd11_init_seed(_ccd11)                                \
105                                                                    \
106  VOID_EXPR_                                                       \
107  (                                                                    \
108      (_ccd11)->size = 0,                                           \
109      (_ccd11)->last = (_ccd11)->vcnt = 0,                          \
110                                                                    \
111      (_ccd11)->head.PRIV = NULL,                                    \
112      (_ccd11)->head.NXT = &((_ccd11)->tail),                      \
113      (_ccd11)->tail.PRIV = &((_ccd11)->head),                     \
114      (_ccd11)->tail.NXT = NULL                                     \
115  )
116
117
118  #define _ccd11_init_info(_ccd11, _start, _ratio, _thrsh)        \
119                                                                    \
120      _cc_11_init_info(_ccd11, _start, _ratio, _thrsh)
121
122
123  #define ccd11_iter_init(_iter, _ccd11)                          \
124                                                                    \
125  VOID_EXPR_                                                       \
126  (                                                                    \
127      (_iter)->curr.node = NULL,                                    \
128      (_iter)->cont = (_ccd11)                                     \
129  )
130
131
132  #define _ccd11_iter_init(_iter, _ccd11, _alloc)                \
133                                                                    \
134      _cc_11_iter_init(_iter, _ccd11, _alloc, ccd11)
135
136
137
138  /* ccd11 destroy */
139
140
141  #define ccd11_free(_ccd11)  cc_11_free(_ccd11)
142
143
144
145  /* ccd11 access */
146
147
148  #define ccd11_front(_ccd11)  ((_ccd11)->head.NXT->val)

```

```

149
150 #define ccdll_back(_ccdll)  ((_ccdll)->tail.PRIV->val)
151
152
153
154 /* ccdll capacity */
155
156
157 #define ccdll_size(_ccdll)  ((_ccdll)->size)
158
159 #define ccdll_empty(_ccdll)  ((_ccdll)->head.NXT == &((_ccdll)->tail) && \
160                               (_ccdll)->tail.PRIV == &((_ccdll)->head))
161
162
163
164 /* ccdll modifiers */
165
166
167 #define ccdll_push_front(_ccdll, _val) _ccdll_push(_ccdll, _val, head, NXT, PRIV)
168
169 #define ccdll_push_back(_ccdll, _val) _ccdll_push(_ccdll, _val, tail, PRIV, NXT)
170
171 #define _ccdll_push(_ccdll, _val, _hdtl_, _pn_1_, _pn_2_) \
172 \
173 STATEMENT_ \
174 ( \
175     _ccdll_push_alloc((_ccdll), _hdtl_, _pn_1_, _pn_2_); \
176 \
177     (_ccdll)->_hdtl_.pn_1->val = (_val); \
178 )
179
180
181 #define ccdll_push_front_alloc(_ccdll) _ccdll_push_alloc(_ccdll, head, NXT, PRIV)
182
183 #define ccdll_push_back_alloc(_ccdll) _ccdll_push_alloc(_ccdll, tail, PRIV, NXT)
184
185 #define _ccdll_push_alloc(_ccdll, _hdtl_, _pn_1_, _pn_2_) \
186 \
187 STATEMENT_ \
188 ( \
189     _node_alloc((_ccdll)->pnode, (_ccdll)); \
190 \
191     (_ccdll)->pnode->pn_1_ = (_ccdll)->_hdtl_.pn_1_; \
192     (_ccdll)->pnode->pn_2_ = &((_ccdll)->_hdtl_); \
193 \
194     (_ccdll)->pnode->PRIV->NXT = (_ccdll)->pnode; \
195     (_ccdll)->pnode->NXT->PRIV = (_ccdll)->pnode; \
196 \
197     (_ccdll)->size++; \
198 )

```

```

199
200
201 #define ccdll_pop_front(_ccdll)  _ccdll_pop(_ccdll, head, NXT)
202
203 #define ccdll_pop_back(_ccdll)   _ccdll_pop(_ccdll, tail, PRV)
204
205 #define _ccdll_pop(_ccdll, _hdtl_, _pn_)
206                                     \
207 STATEMENT_                           \
208 (                                     \
209     if (ccdll_empty((_ccdll))) break; \
210                                     \
211     (_ccdll)->pnode = (_ccdll)->_hdtl_.pn_; \
212                                     \
213     (_ccdll)->pnode->PRV->NXT = (_ccdll)->pnode->NXT; \
214     (_ccdll)->pnode->NXT->PRV = (_ccdll)->pnode->PRV; \
215                                     \
216     _node_clear((_ccdll)->pnode, (_ccdll)); \
217                                     \
218     (_ccdll)->size--; \
219 )
220
221
222 #define ccdll_insert(_iter, _val)
223                                     \
224 STATEMENT_                           \
225 (                                     \
226     if (ccdll_iter_at_head((_iter))) break; \
227                                     \
228     _node_alloc((_iter)->cont->pnode, (_iter)->cont); \
229                                     \
230     (_iter)->cont->pnode->val = (_val); \
231                                     \
232     (_iter)->cont->pnode->PRV = (_iter)->curr.node->PRV; \
233     (_iter)->cont->pnode->NXT = (_iter)->curr.node; \
234                                     \
235     (_iter)->curr.node->PRV->NXT = (_iter)->cont->pnode; \
236     (_iter)->curr.node->PRV      = (_iter)->cont->pnode; \
237     (_iter)->curr.node          = (_iter)->cont->pnode; \
238                                     \
239     (_iter)->cont->size++; \
240 )
241
242
243 #define ccdll_erase(_iter)
244                                     \
245 STATEMENT_                           \
246 (                                     \
247     if (ccdll_iter_at_head((_iter)) || ccdll_iter_at_tail((_iter))) break; \
248                                     \

```

```

249     (_iter)->curr.node->PRV->NXT = (_iter)->curr.node->NXT;           \
250     (_iter)->curr.node->NXT->PRV = (_iter)->curr.node->PRV;           \
251                                                                 \
252     (_iter)->curr.node->PRV = (_iter)->curr.node->NXT;               \
253     _node_clear((_iter)->curr.node, (_iter)->cont);                 \
254     (_iter)->curr.node      = (_iter)->curr.node->PRV;               \
255                                                                 \
256     (_iter)->cont->size--;                                           \
257 )                                                                 \
258                                                                 \
259                                                                 \
260 #define ccdll_swap(_ccdll_a, _ccdll_b)                             \
261                                                                 \
262 STATEMENT_                                                           \
263 (                                                                     \
264     XOR2_SWAP((_ccdll_a), (_ccdll_b));                             \
265 )                                                                     \
266                                                                 \
267                                                                 \
268 #define ccdll_resize(_ccdll, _items, _val)                         \
269                                                                 \
270 STATEMENT_                                                           \
271 (                                                                     \
272     int _size = ccdll_size((_ccdll)) - (_items);                  \
273                                                                 \
274     if (_size > 0) while(_size--) ccdll_pop_back((_ccdll));        \
275     else if (_size < 0) while(_size++) ccdll_push_back((_ccdll), (_val)); \
276 )                                                                     \
277                                                                 \
278                                                                 \
279 #define ccdll_clear(_ccdll)                                         \
280                                                                 \
281 STATEMENT_                                                           \
282 (                                                                     \
283     while (!(ccdll_empty((_ccdll)))) ccdll_pop_back((_ccdll));    \
284 )                                                                     \
285                                                                 \
286                                                                 \
287                                                                 \
288 /* ccdll operations */                                             \
289                                                                 \
290                                                                 \
291 #define ccdll_move(_iter_p, _iter_i)                               \
292                                                                 \
293 STATEMENT_                                                           \
294 (                                                                     \
295     if (_unlikely((_iter_i)->curr.node == (_iter_p)->curr.node->PRV)) break; \
296                                                                 \
297     if (_unlikely(ccdll_iter_at_head((_iter_p)) ||                  \
298                 ccdll_iter_at_head((_iter_i)) ||                  \

```

```

299         ccdll_iter_at_tail((_iter_i))) break; \
300 \
301     _ccdll_move((_iter_p), (_iter_i)); \
302 \
303     (_iter_i)->cont->size--; \
304     (_iter_i)->cont = (_iter_p)->cont; \
305     (_iter_i)->cont->size++; \
306 )
307
308 #define _ccdll_move(_iter_p, _iter_i) \
309 \
310 STATEMENT_ \
311 ( \
312     (_iter_i)->curr.node->PRV->NXT = (_iter_i)->curr.node->NXT; \
313     (_iter_i)->curr.node->NXT->PRV = (_iter_i)->curr.node->PRV; \
314 \
315     (_iter_i)->curr.node->PRV      = (_iter_p)->curr.node->PRV; \
316     (_iter_i)->curr.node->NXT      = (_iter_p)->curr.node; \
317 \
318     (_iter_p)->curr.node->PRV      = (_iter_i)->curr.node; \
319     (_iter_i)->curr.node->PRV->NXT = (_iter_i)->curr.node; \
320 )
321
322
323 #define ccdll_move_begin(_iter_a, _iter_b) \
324 \
325 STATEMENT_ \
326 ( \
327     ccdll_iter_begin((_iter_a)); \
328     ccdll_iter_begin((_iter_b)); \
329 \
330     ccdll_move((_iter_a), (_iter_b)); \
331 )
332
333
334 #define ccdll_move_range(_iter_p, _iter_l, _iter_r) \
335 \
336     ccdll_move_range_extd(_iter_p, _iter_l, _iter_r, -1)
337
338 #define ccdll_move_range_extd(_iter_p, _iter_l, _iter_r, _dist) /* [l, r) */ \
339 \
340 STATEMENT_ \
341 ( \
342     if (_unlikely(ccdll_iter_at_head ((_iter_p)) || \
343         ccdll_iter_at_head ((_iter_l)))) break; \
344 \
345     if (_unlikely((_iter_l)->curr.node == (_iter_r)->curr.node)) break; \
346 \
347     if (_unlikely((_iter_l)->cont != (_iter_r)->cont)) break; \
348

```

```

349     if (_unlikely((_iter_p)->cont != (_iter_l)->cont))
350     {
351         int _dist_m = (_dist);
352
353         if (_dist_m < 0)
354             ccdll_iter_distance((_iter_l), (_iter_r), &_amp;_dist_m);
355
356         if (_dist_m <= 0) break;
357
358         (_iter_p)->cont->size += _dist_m;
359         (_iter_l)->cont->size -= _dist_m;
360         (_iter_l)->cont = (_iter_p)->cont;
361     }
362
363     void *_pbup = (_iter_r)->curr.node;
364
365     (_iter_r)->curr.node = (_iter_r)->curr.node->PRV;
366
367     (_iter_l)->curr.node->PRV->NXT = (_iter_r)->curr.node->NXT;
368     (_iter_r)->curr.node->NXT->PRV = (_iter_l)->curr.node->PRV;
369
370     (_iter_l)->curr.node->PRV      = (_iter_p)->curr.node->PRV;
371     (_iter_p)->curr.node->PRV->NXT = (_iter_l)->curr.node;
372
373     (_iter_r)->curr.node->NXT      = (_iter_p)->curr.node;
374     (_iter_p)->curr.node->PRV      = (_iter_r)->curr.node;
375
376     (_iter_r)->curr.node = _pbup;
377 )
378
379
380 #define ccdll_merge(_ccdll_d, _ccdll_s)
381
382     ccdll_merge_extd(_ccdll_d, _ccdll_s, DLEQ)
383
384 #define ccdll_merge_extd(_ccdll_d, _ccdll_s, _leq)
385
386     cc_ll_merge_extd(_ccdll_d, _ccdll_s, _leq, ccdll, )
387
388 #define _ccdll_merge_extd(_iter_l, _iter_m, _iter_r, _leq)
389
390     _cc_ll_merge_extd(_iter_l, _iter_m, _iter_r, _leq, ccdll, )
391
392
393 #define ccdll_merge_range(_iter_l, _iter_m, _iter_r)
394
395     ccdll_merge_range_extd(_iter_l, _iter_m, _iter_r, DLEQ)
396
397 #define ccdll_merge_range_extd(_iter_l, _iter_m, _iter_r, _leq)
398

```

```

399         cc_ll_merge_range_extd(_iter_l, _iter_m, _iter_r, _leq, ccdll, )
400
401 #define _ccdll_merge_range_extd(_iter_l, _iter_m, _iter_r, _iter_x, _leq)      \
402                                                                                   \
403         _cc_ll_merge_range_extd(_iter_l, _iter_m, _iter_r, _iter_x, _leq, ccdll)
404
405
406 #define ccdll_sort(_ccdll)                                                       \
407                                                                                   \
408         ccdll_sort_extd(_ccdll, DLEQ)
409
410 #define ccdll_sort_extd(_ccdll, _leq)                                           \
411                                                                                   \
412         cc_ll_sort_extd(_ccdll, _leq, ccdll, )
413
414 #define _ccdll_sort_extd(_ccdll, _carry, _pbuck, _iter_a, _iter_b, _leq)      \
415                                                                                   \
416         _cc_ll_sort_extd(_ccdll, _carry, _pbuck, _iter_a, _iter_b, _leq, ccdll,)
417
418
419
420 /* ccdll comparator */
421
422
423 #define ccdll_comp_leq(_iter_a, _iter_b)      (DREF((_iter_a)) <=          \
424                                                DREF((_iter_b)))
425
426 #define ccdll_comp_leq_prev(_iter_a, _iter_b) (DREF_PREV((_iter_a)) <=    \
427                                                DREF_PREV((_iter_b)))
428
429 #define ccdll_comp_leq_next(_iter_a, _iter_b) (DREF_NEXT((_iter_a)) <=    \
430                                                DREF_NEXT((_iter_b)))
431
432 #define ccdll_comp_geq(_iter_a, _iter_b)      (DREF((_iter_a)) >=          \
433                                                DREF((_iter_b)))
434
435 #define ccdll_comp_geq_prev(_iter_a, _iter_b) (DREF_PREV((_iter_a)) >=    \
436                                                DREF_PREV((_iter_b)))
437
438 #define ccdll_comp_geq_next(_iter_a, _iter_b) (DREF_NEXT((_iter_a)) >=    \
439                                                DREF_NEXT((_iter_b)))
440
441
442
443 /* ccdll iterators */
444
445
446 #define ccdll_iter_copy(_iter_dst, _iter_src)                                   \
447                                                                                   \
448 VOID_EXPR_

```



```

449 ( \
450     *(_iter_dst) = *(_iter_src) \
451 )
452
453
454 #define ccdll_iter_head(_iter) \
455 \
456 VOID_EXPR_ \
457 ( \
458     (_iter)->curr.node = &((_iter)->cont->head) \
459 )
460
461
462 #define ccdll_iter_tail(_iter) \
463 \
464 VOID_EXPR_ \
465 ( \
466     (_iter)->curr.node = &((_iter)->cont->tail) \
467 )
468
469
470 #define ccdll_iter_begin(_iter) \
471 \
472 VOID_EXPR_ \
473 ( \
474     (_iter)->curr.node = ((_iter)->cont->head.NXT) \
475 )
476
477
478 #define ccdll_iter_end(_iter) \
479 \
480 VOID_EXPR_ \
481 ( \
482     (_iter)->curr.node = ((_iter)->cont->tail.PRIV) \
483 )
484
485
486 #define ccdll_iter_at_head(_iter) ( (_iter)->curr.node->PRIV == NULL )
487
488 #define ccdll_iter_at_tail(_iter) ( (_iter)->curr.node->NXT == NULL )
489
490 #define ccdll_iter_at_begin(_iter) ( (_iter)->curr.node->PRIV == \
491                                     &((_iter)->cont->head) )
492
493 #define ccdll_iter_at_end(_iter) ( (_iter)->curr.node->NXT == \
494                                     &((_iter)->cont->tail) )
495
496
497 #define ccdll_iter_incr(_iter) \
498 ( \

```

```

499     ccdll_iter_at_tail((_iter)) ? (NULL) :                                \
500     ((_iter)->curr.node = (_iter)->curr.node->NXT)->NXT                \
501 )
502
503
504 #define ccdll_iter_decr(_iter)                                           \
505 (                                                                           \
506     ccdll_iter_at_head((_iter)) ? (NULL) :                               \
507     ((_iter)->curr.node = (_iter)->curr.node->PRV)->PRV                 \
508 )
509
510
511 #define ccdll_iter_advance(_iter, _diff)                                  \
512                                     \
513 STATEMENT_                                                                \
514 (                                                                           \
515     int _len = (_diff);                                                    \
516                                     \
517     if (_len > 0) while (ccdll_iter_incr((_iter)) && --_len);            \
518     else if (_len < 0) while (ccdll_iter_decr((_iter)) && ++_len);        \
519 )
520
521
522 #define ccdll_iter_distance(_iter_a, _iter_b, _pdist)                    \
523                                     \
524     cc_ll_iter_distance(_iter_a, _iter_b, _pdist, ccdll)                 \
525
526
527
528 /* ccdll traversor */
529
530
531 #define CCDLL_INCR(_iter)  CC_LL_INCR(_iter, ccdll)
532
533 #ifndef CC_STRICT
534
535 #define CCDLL_INCR_AUTO(_pval, _ccdll)                                    \
536                                     \
537     CCDLL_INCR_EXTD(_pval, _ccdll, (void)0)
538
539 #define CCDLL_INCR_EXTD(_pval, _ccdll, ...)                               \
540                                     \
541     CC_LL_INCR_EXTD(_pval, _ccdll, ccdll, __VA_ARGS__)
542
543 #endif // CC_STRICT
544
545
546 #define CCDLL_DECR(_iter)  CC_LL_DECR(_iter, ccdll)
547
548 #ifndef CC_STRICT

```

```

549
550 #define CCDLL_DECR_AUTO(_pval, _ccd11) \
551 \
552         CCDLL_DECR_EXTD(_pval, _ccd11, (void)0)
553
554 #define CCDLL_DECR_EXTD(_pval, _ccd11, ...) \
555 \
556         CC_LL_DECR_EXTD(_pval, _ccd11, ccd11, __VA_ARGS__)
557
558 #endif // CC_STRICT
559
560
561
562 #endif

```

1.1.4 list/ccsll.h

```
1  #ifndef OPENG3_LIST_CCSSL_H
2  #define OPENG3_LIST_CCSSL_H
3
4  #include "base.h"
5  #include "../base/pool.h"
6  #include "../base/misc.h"
7  #include "../base/snym.h"
8
9  #include <math.h>
10 #include <stddef.h>
11 #include <stdint.h>
12 #include <string.h>
13
14
15 /* ccsll create */
16
17
18 #define ccsll(elem_t) \
19 \
20     ccsll_extd(elem_t, 1, NORMAL)
21
22 #define ccsll_pckd(elem_t) \
23 \
24     ccsll_extd(elem_t, 1, PACKED)
25
26 #define ccsll_extd(elem_t, _n_iter, _ALIGN_) \
27 \
28     typedef ccsll_struct_extd(elem_t, _n_iter, _ALIGN_) *CCSLL; CCSLL
29
30 #define ccsll_type(elem_t) \
31 \
32     typedef ccsll_struct_extd(elem_t, 1, NORMAL) *
33
34
35 #define ccsll_struct(elem_t) \
36 \
37     ccsll_struct_extd(elem_t, 1, NORMAL)
38
39 #define ccsll_struct_pckd(elem_t) \
40 \
41     ccsll_struct_extd(elem_t, 1, PACKED)
42
43 #define ccsll_struct_extd(elem_t, _n_iter, _ALIGN_) \
44 \
45     struct CCSLL_CONT \
46     { \
47         int size, last, vcnt; /* size and node record */ \
48         int start, ratio, thrsh; /* block increment info */ \

```

```

49                                     \
50 struct CCSLL_NODE                                     \
51 { struct CCSLL_NODE *lnk[1];                                     \
52   elem_t val;                                           /* val with a next link */ \
53 } *avsp, *pnode, head, tail, swap;                     /* available space list */ \
54                                     \
55 struct CCSLL_BLK                                     \
56 { struct CCSLL_BLK *bprv, *bnxt;                         /* points to prev block */ \
57   int ncnt;                                             /* the item of the node */ \
58   PRAGMA_##_ALIGN_##_BGN                               /* packed pragma starts */ \
59   struct CCSLL_NODE nodes[];                          /* node structure array */ \
60   PRAGMA_##_ALIGN_##_END                               /* the pragma ends here */ \
61 } *pool, *pblock;                                     /* points to 1-st block */ \
62                                     \
63 struct CCSLL_ITER                                     \
64 { struct CCSLL_PTRS                                     \
65   { struct CCSLL_NODE *node;                             \
66     } curr;                                             /* points to curr node */ \
67   struct CCSLL_CONT *cont;                             /* points to ccsll body */ \
68 } (*itarr)[_n_iter], *_iter, **_it;                  \
69                                     \
70 struct CCSLL_CONT *_co;                               /* internal use _it _co */ \
71                                     \
72 int _it_base, _it_limit;                               \
73 int _co_base, _co_limit;                               \
74 }
75
76
77
78 /* ccsll initialize */
79
80
81 #define ccsll_init(_ccsll)                                     \
82                                     \
83     ccsll_init_extd(_ccsll,      16,      2,  65536)
84
85 #define ccsll_init_extd(_ccsll, _start, _ratio, _thrsh)      \
86                                     \
87     cc_ll_init_extd(_ccsll, _start, _ratio, _thrsh, ccsll)
88
89
90 #define _ccsll_init(_ccsll_dst, _ccsll_src, _alloc)          \
91                                     \
92     _ccsll_init_extd(_ccsll_dst, -1,      -1,      -1, _alloc)
93
94 #define _ccsll_init_extd(_ccsll, _start, _ratio, _thrsh, _alloc) \
95                                     \
96     _cc_ll_init_extd(_ccsll, _start, _ratio, _thrsh, _alloc, ccsll)
97
98

```

```

99  #define _ccsll_init_core(_ccsll)                                \
100                                                                    \
101      _cc_ll_init_core(_ccsll, ccsll)
102
103
104  #define _ccsll_init_seed(_ccsll)                                \
105                                                                    \
106  VOID_EXPR_                                                       \
107  (                                                                    \
108      (_ccsll)->size = 0,                                           \
109      (_ccsll)->last = (_ccsll)->vcnt = 0,                         \
110                                                                    \
111      (_ccsll)->head.NXT = &((_ccsll)->tail),                     \
112      (_ccsll)->tail.NXT = NULL                                    \
113  )
114
115
116  #define _ccsll_init_info(_ccsll, _start, _ratio, _thrsh)        \
117                                                                    \
118      _cc_ll_init_info(_ccsll, _start, _ratio, _thrsh)
119
120
121  #define ccsll_iter_init(_iter, _ccsll)                          \
122                                                                    \
123  VOID_EXPR_                                                       \
124  (                                                                    \
125      (_iter)->curr.node = NULL,                                    \
126      (_iter)->cont = (_ccsll)                                     \
127  )
128
129
130  #define _ccsll_iter_init(_iter, _ccsll, _alloc)                \
131                                                                    \
132      _cc_ll_iter_init(_iter, _ccsll, _alloc, ccsll)
133
134
135
136  /* ccsll destroy */
137
138
139  #define ccsll_free(_ccsll)  cc_ll_free(_ccsll)
140
141
142
143  /* ccsll access */
144
145
146  #define ccsll_front(_ccsll)  ((_ccsll)->head.NXT->val)
147
148

```

```

149
150  /* ccsll capacity */
151
152
153  #define ccsll_size(_ccsll)    ((_ccsll)->size)
154
155  #define ccsll_empty(_ccsll)  ((_ccsll)->head.NXT == &((_ccsll)->tail))
156
157
158
159  /* ccsll modifiers */
160
161
162  #define ccsll_push_front(_ccsll, _val)                                \
163                                                                           \
164  STATEMENT_                                                             \
165  (                                                                           \
166      ccsll_push_front_alloc((_ccsll));                                   \
167                                                                           \
168      (_ccsll)->head.NXT->val = (_val);                                   \
169  )
170
171
172  #define ccsll_push_front_alloc(_ccsll)                                \
173                                                                           \
174  STATEMENT_                                                             \
175  (                                                                           \
176      _node_alloc((_ccsll)->pnode, (_ccsll));                           \
177                                                                           \
178      (_ccsll)->pnode->NXT = (_ccsll)->head.NXT;                         \
179      (_ccsll)->head.NXT  = (_ccsll)->pnode;                             \
180                                                                           \
181      (_ccsll)->size++;                                                  \
182  )
183
184
185  #define ccsll_pop_front(_ccsll)                                        \
186                                                                           \
187  STATEMENT_                                                             \
188  (                                                                           \
189      if (ccsll_empty((_ccsll))) break;                                   \
190                                                                           \
191      (_ccsll)->pnode    = (_ccsll)->head.NXT;                           \
192      (_ccsll)->head.NXT = (_ccsll)->pnode->NXT;                         \
193                                                                           \
194      _node_clear((_ccsll)->pnode, (_ccsll));                             \
195                                                                           \
196      (_ccsll)->size--;                                                  \
197  )
198

```

```

199
200 #define ccsl_insert(_iter, _val) \
201 \
202 STATEMENT_ \
203 ( \
204     if (ccsl_iter_at_head((_iter))) break; \
205 \
206     _node_alloc((_iter)->cont->pnode, (_iter)->cont); \
207 \
208     /* TODO */ \
209 \
210     (_iter)->cont->size++; \
211 ) \
212 \
213
214 #define ccsl_erase(_iter) \
215 \
216 STATEMENT_ \
217 ( \
218     if (ccsl_iter_at_head((_iter)) || ccsl_iter_at_tail((_iter))) break; \
219 \
220     /* TODO */ \
221 \
222     _node_clear((_iter)->curr.node, (_iter)->cont); \
223 \
224     (_iter)->cont->size--; \
225 ) \
226 \
227
228 #define ccsl_swap(_ccsl_a, _ccsl_b) \
229 \
230 STATEMENT_ \
231 ( \
232     XOR2_SWAP((_ccsl_a), (_ccsl_b)); \
233 ) \
234 \
235
236 #define ccsl_resize(_ccsl, _items, _val) \
237 \
238 STATEMENT_ \
239 ( \
240     int _size = ccsl_size((_ccsl)) - (_items); \
241 \
242     if (_size > 0) while(_size--) ccsl_pop_front((_ccsl)); \
243     else if (_size < 0) while(_size++) ccsl_push_front((_ccsl), (_val)); \
244 ) \
245 \
246
247 #define ccsl_clear(_ccsl) \
248 \

```



```

249 STATEMENT_ \
250 ( \
251     while (!(ccsll_empty((_ccsll))))      ccsll_pop_front((_ccsll)); \
252 ) \
253 \
254 \
255 \
256 /* ccsll operations */
257 \
258 \
259 #define ccsll_move(_iter_p, _iter_i) \
260 \
261 STATEMENT_ \
262 ( \
263     if (_unlikely((_iter_i)->curr.node == (_iter_p)->curr.node)) break; \
264 \
265     if (_unlikely(ccsll_iter_at_tail((_iter_p)) || \
266                 ccsll_iter_at_end((_iter_i)) || \
267                 ccsll_iter_at_tail((_iter_i)))) break; \
268 \
269     _ccsll_move((_iter_p), (_iter_i)); \
270 \
271     (_iter_p)->cont->size++; \
272     (_iter_i)->cont->size--; \
273 ) \
274 \
275 #define _ccsll_move(_iter_p, _iter_i) \
276 \
277 STATEMENT_ \
278 ( \
279     void *_pbup = (_iter_i)->curr.node->NXT->NXT; \
280 \
281     (_iter_i)->curr.node->NXT->NXT = (_iter_p)->curr.node->NXT; \
282     (_iter_p)->curr.node->NXT      = (_iter_i)->curr.node->NXT; \
283     (_iter_i)->curr.node->NXT      = _pbup; \
284 ) \
285 \
286 \
287 #define ccsll_move_begin(_iter_a, _iter_b) \
288 \
289 STATEMENT_ \
290 ( \
291     ccsll_iter_head((_iter_a)); \
292     ccsll_iter_head((_iter_b)); \
293 \
294     ccsll_move((_iter_a), (_iter_b)); \
295 ) \
296 \
297 \
298 #define ccsll_move_range(_iter_p, _iter_l, _iter_r) \

```

```

299                                     \
300         ccsl1_move_range_extd(_iter_p, _iter_l, _iter_r,    -1)
301
302 #define ccsl1_move_range_extd(_iter_p, _iter_l, _iter_r, _dist) /* (l, r] */ \
303
304 STATEMENT_ \
305 ( \
306     if (_unlikely(ccsl1_iter_at_tail((_iter_p)) || \
307                   ccsl1_iter_at_tail((_iter_r)))) break; \
308
309     if (_unlikely((_iter_l)->curr.node == (_iter_r)->curr.node)) break; \
310
311     if (_unlikely((_iter_l)->cont != (_iter_r)->cont)) break; \
312
313     if (_unlikely((_iter_p)->cont != (_iter_r)->cont)) \
314     { \
315         int _dist_m = (_dist); \
316
317         if (_dist_m < 0) \
318             ccsl1_iter_distance((_iter_l), (_iter_r), &_dist_m); \
319
320         if (_dist_m <= 0) break; \
321
322         (_iter_p)->cont->size += _dist_m; \
323         (_iter_r)->cont->size -= _dist_m; \
324         (_iter_r)->cont = (_iter_p)->cont; \
325     } \
326
327     void *_pbup = (_iter_p)->curr.node->NXT; \
328
329     (_iter_p)->curr.node->NXT = (_iter_l)->curr.node->NXT; \
330     (_iter_l)->curr.node->NXT = (_iter_r)->curr.node->NXT; \
331     (_iter_r)->curr.node->NXT = _pbup; \
332 )
333
334
335 #define ccsl1_merge(_ccsl1_d, _ccsl1_s) \
336
337     ccsl1_merge_extd(_ccsl1_d, _ccsl1_s, SLEQ_NEXT)
338
339 #define ccsl1_merge_extd(_ccsl1_d, _ccsl1_s, _leq) \
340
341     cc_ll_merge_extd(_ccsl1_d, _ccsl1_s, _leq, ccsl1, )
342
343 #define _ccsl1_merge_extd(_iter_l, _iter_m, _iter_r, _leq) \
344
345 STATEMENT_ \
346 ( \
347     if (_unlikely((_iter_l)->cont == (_iter_m)->cont || \
348                   (_iter_l)->cont == (_iter_r)->cont ||

```

```

349         (_iter_m)->cont != (_iter_r)->cont)) break; \
350 \
351     ccsl1_iter_head((_iter_l)); \
352     ccsl1_iter_head((_iter_m)); \
353     ccsl1_iter_head((_iter_r)); \
354 \
355     for (register int _len = 0; ; _len = 0) \
356     { \
357         while (!(ccsl1_iter_at_end((_iter_l))) && !_leq((_iter_l), (_iter_m))) \
358             ((void)ccsl1_iter_incr((_iter_l))); \
359 \
360         while (!(ccsl1_iter_at_end((_iter_r))) && \
361             (ccsl1_iter_at_end((_iter_l)) || !_leq((_iter_l), (_iter_r)))) \
362             ((void)ccsl1_iter_incr((_iter_r)), ++_len); \
363 \
364         ccsl1_move_range_extd((_iter_l), (_iter_m), (_iter_r), _len); \
365 \
366         if (ccsl1_iter_at_end((_iter_m))) break; \
367 \
368         ccsl1_iter_copy((_iter_l), (_iter_r)); \
369         ccsl1_iter_init((_iter_r), (_iter_m)->cont); \
370         ccsl1_iter_head((_iter_r)); \
371     } \
372 ) \
373 \
374 \
375 #define ccsl1_sort(_ccsl1) \
376 \
377     ccsl1_sort_extd(_ccsl1, SLEQ_NEXT) \
378 \
379 #define ccsl1_sort_extd(_ccsl1, _leq) \
380 \
381     cc_ll_sort_extd(_ccsl1, _leq, ccsl1, ) \
382 \
383 #define _ccsl1_sort_extd(_ccsl1, _carry, _pbuck, _iter_a, _iter_b, _leq) \
384 \
385     _cc_ll_sort_extd(_ccsl1, _carry, _pbuck, _iter_a, _iter_b, _leq, ccsl1,) \
386 \
387 \
388 \
389 /* ccsl1 comparator */ \
390 \
391 \
392 #define ccsl1_comp_leq(_iter_a, _iter_b) (SREF((_iter_a)) <= \
393     SREF((_iter_b))) \
394 \
395 #define ccsl1_comp_leq_next(_iter_a, _iter_b) (SREF_NEXT((_iter_a)) <= \
396     SREF_NEXT((_iter_b))) \
397 \
398 #define ccsl1_comp_geq(_iter_a, _iter_b) (SREF((_iter_a)) >= \

```

```

399                                     SREF((_iter_b)))
400
401 #define ccsl_comp_geq_next(_iter_a, _iter_b) (SREF_NEXT((_iter_a)) >= \
402                                             SREF_NEXT((_iter_b)))
403
404
405
406 /* ccsl iterators */
407
408
409 #define ccsl_iter_copy(_iter_dst, _iter_src) \
410 \
411 VOID_EXPR_ \
412 ( \
413     *(_iter_dst) = *(_iter_src) \
414 )
415
416
417 #define ccsl_iter_head(_iter) \
418 \
419 VOID_EXPR_ \
420 ( \
421     (_iter)->curr.node = &((_iter)->cont->head) \
422 )
423
424
425 #define ccsl_iter_tail(_iter) \
426 \
427 VOID_EXPR_ \
428 ( \
429     (_iter)->curr.node = &((_iter)->cont->tail) \
430 )
431
432
433 #define ccsl_iter_begin(_iter) \
434 \
435 VOID_EXPR_ \
436 ( \
437     (_iter)->curr.node = ((_iter)->cont->head.NXT) \
438 )
439
440
441 #define ccsl_iter_at_head(_iter) ( (_iter)->curr.node == \
442                                     &((_iter)->cont->head) )
443
444 #define ccsl_iter_at_tail(_iter) ( (_iter)->curr.node == \
445                                     &((_iter)->cont->tail) )
446
447 #define ccsl_iter_at_begin(_iter) ( (_iter)->curr.node == \
448                                     (_iter)->cont->head.NXT )

```

```

449
450 #define ccsl_iter_at_end(_iter)      ( (_iter)->curr.node->NXT ==      \
451                                     &((_iter)->cont->tail))
452
453
454 #define ccsl_iter_incr(_iter)        \
455 (                                     \
456     ccsl_iter_at_tail((_iter)) ? (NULL) :      \
457     ((_iter)->curr.node = (_iter)->curr.node->NXT->NXT      \
458 )
459
460
461 #define ccsl_iter_advance(_iter, _diff)        \
462                                                 \
463 STATEMENT_                                     \
464 (                                               \
465     int _len = (_diff);                        \
466                                                 \
467     if (_len > 0) while (ccsl_iter_incr((_iter)) && --_len);      \
468 )
469
470
471 #define ccsl_iter_distance(_iter_a, _iter_b, _pdist)        \
472                                                             \
473     cc_ll_iter_distance_positive(_iter_a, _iter_b, _pdist, ccsl)
474
475
476
477 /* ccsl traversor */
478
479
480 #define CCSLL_INCR(_iter)  CC_LL_INCR(_iter, ccsl)
481
482 #ifndef CC_STRICT
483
484 #define CCSLL_INCR_AUTO(_pval, _ccsl)        \
485                                                 \
486     CCSLL_INCR_EXTD(_pval, _ccsl, (void)0)
487
488 #define CCSLL_INCR_EXTD(_pval, _ccsl, ...)    \
489                                                 \
490     CC_LL_INCR_EXTD(_pval, _ccsl, ccsl, __VA_ARGS__)
491
492 #endif // CC_STRICT
493
494
495
496 #endif

```

1.1.5 list/extd-base.h

```

1  #ifndef OPENG3_LIST_EXTD_BASE_H
2  #define OPENG3_LIST_EXTD_BASE_H
3
4
5  /* cc_ll integrity */
6
7
8  #define cc_ll_is_sorted_extd(_cc_ll, _leq, _ptrue, _ll_) \
9 \
10 STATEMENT_ \
11 ( \
12     (*(_ptrue)) = 1; \
13 \
14     if (_ll_##_size((_cc_ll)) <= 1) break; \
15 \
16     _it_init((_cc_ll), 2, _base_s3, _ll_); \
17 \
18     _ll_##_iter_head (_it((_cc_ll), _base_s3, 0)); \
19     _ll_##_iter_begin(_it((_cc_ll), _base_s3, 1)); \
20 \
21     while (1) \
22     { \
23         (void)_ll_##_iter_incr(_it((_cc_ll), _base_s3, 0)); \
24         (void)_ll_##_iter_incr(_it((_cc_ll), _base_s3, 1)); \
25 \
26         if (!(_leq(_it((_cc_ll), _base_s3, 0), _it((_cc_ll), _base_s3, 1)))) \
27         { \
28             (*(_ptrue)) = 0; break; \
29         } \
30 \
31         if (_ll_##_iter_at_end(_it((_cc_ll), _base_s3, 1))) break; \
32     } \
33 \
34     _it_clear((_cc_ll), 2); \
35 )
36
37
38 #define cc_ll_is_robust(_cc_ll, _ptrue, _ll_, _LL_) /* INCOMPLETE */ \
39 \
40 STATEMENT_ \
41 ( \
42     int _size = 0; \
43 \
44     _LL_##_INCR((_cc_ll)->_iter) _size++; \
45 \
46     *(_ptrue) = !((_ll_##_size((_cc_ll)) == _size); \
47 )
48

```

49

50

51 #endif

1.1.6 list/extd-ccxll.h

```
1  #ifndef OPENG3_LIST_EXTD_CCXLL_H
2  #define OPENG3_LIST_EXTD_CCXLL_H
3
4  #include "ccxll.h"
5  #include "extd-base.h"
6  #include "../vect/array.h"
7
8
9  /* ccxll operations extended */
10
11
12  #define ccxll_sort_destruct(_ccxll) /* SIZE OF MEMORY POOL BLOCKS MATTERS */ \
13                                     \
14         ccxll_sort_destruct_extd(_ccxll, XLEQ, ACMP)
15
16  #define ccxll_sort_destruct_extd(_ccxll, _leq, _cmp) \
17                                     \
18  STATEMENT_ \
19  ( \
20         int _sbup = ccxll_size((_ccxll)); \
21                                     \
22         (_ccxll)->pblock = (_ccxll)->pool; \
23                                     \
24         while ((_ccxll)->pblock != NULL) \
25         { \
26                 int _lo = ((_ccxll)->pblock->bnxt == NULL) ? (_ccxll)->vcnt : 0; \
27                 int _hi = ((_ccxll)->pblock->ncnt - 1); \
28                                     \
29                 array_sort((_ccxll)->pblock->nodes, &(_ccxll)->swap, _lo, _hi, _cmp); \
30                                     \
31                 (_ccxll)->pool = (_ccxll)->pblock; \
32                 (_ccxll)->pblock = (_ccxll)->pblock->bprv; \
33         } \
34                                     \
35         _ccxll_init_seed((_ccxll)); \
36                                     \
37         while (_sbup-- > 0) \
38                 ccxll_push_front_alloc((_ccxll)); \
39                                     \
40         ccxll_block_merge_extd((_ccxll), _leq); \
41 ) \
42
43
44  #define ccxll_block_merge_extd(_ccxll, _leq) \
45                                     \
46  STATEMENT_ \
47  ( \
48         _it_init((_ccxll), 3, _base_s1, ccxll); \
49                                     \
```



```

49                                     \
50     _ccxll_block_merge_extd((_ccxll), _it((_ccxll), _base_s1, 0),      \
51                                     _it((_ccxll), _base_s1, 1),          \
52                                     _it((_ccxll), _base_s1, 2), _leq);    \
53                                     \
54     _it_clear((_ccxll), 3);                                           \
55 )                                                                       \
56
57
58 #define _ccxll_block_merge_extd(_ccxll, _iter_l, _iter_m, _iter_r, _leq) \
59                                     \
60 STATEMENT_                                                                \
61 (                                                                           \
62     if ((_ccxll)->pool == NULL) break;                                   \
63                                     \
64     (_ccxll)->pblock = (_ccxll)->pool;                                   \
65                                     \
66     ccxll_iter_begin((_iter_r));                                         \
67                                     \
68     while ((_ccxll)->pblock != NULL)                                     \
69     {                                                                     \
70         ccxll_iter_begin((_iter_l));                                     \
71         ccxll_iter_copy ((_iter_m), (_iter_r));                         \
72                                     \
73         _ccxll_iter_block((_ccxll)->pblock->bprv, (_iter_r));           \
74         ccxll_merge_range_extd((_iter_l), (_iter_m), (_iter_r), _leq); \
75                                     \
76         (_ccxll)->pblock = (_ccxll)->pblock->bprv;                     \
77     }                                                                     \
78 )                                                                           \
79
80
81 #define _ccxll_iter_block(_pblock, _iter)                                \
82                                     \
83 STATEMENT_                                                                \
84 (                                                                           \
85     if ((_pblock) == NULL)                                               \
86         ccxll_iter_tail((_iter));                                       \
87     else if ((_pblock)->bnxt == NULL)                                    \
88         ccxll_iter_begin((_iter));                                       \
89     else                                                                    \
90     {                                                                     \
91         (_iter)->curr.XOR = &(_pblock)->nodes[0];                     \
92         (_iter)->prev.XOR = &(_pblock)->bnxt->nodes[(_pblock)->bnxt->ncnt - 1]; \
93         (_iter)->next.XOR = XOR2((_iter)->curr.node->XOR, (_iter)->prev.XOR); \
94     }                                                                     \
95 )                                                                           \
96
97
98

```

```

99  /* ccxll integrity */
100
101
102  #define ccxll_is_sorted(_ccxll, _ptrue) \
103                                          \
104          ccxll_is_sorted_extd(_ccxll, XLEQ, _ptrue)
105
106  #define ccxll_is_sorted_extd(_ccxll, _leq, _ptrue) \
107                                                  \
108          cc_ll_is_sorted_extd(_ccxll, _leq, _ptrue, ccxll)
109
110
111  #define ccxll_is_robust(_ccxll, _ptrue) \
112                                          \
113          cc_ll_is_robust(_ccxll, _ptrue, ccxll, CCXLL)
114
115
116
117  #endif

```

1.1.7 list/extd-ccd11.h

```

1  #ifndef OPENG3_LIST_EXTD_CCD11_H
2  #define OPENG3_LIST_EXTD_CCD11_H
3
4  #include "ccd11.h"
5  #include "extd-base.h"
6  #include "extd-ccs11.h"
7
8
9  /* ccd11 operations extended */
10
11
12 #define ccd11_sort_prefetch(_ccd11) \
13 \
14     ccd11_sort_prefetch_extd(_ccd11, SLEQ_NEXT)
15
16 #define ccd11_sort_prefetch_extd(_ccd11, _leq) \
17 \
18     cc_ll_sort_extd      (_ccd11, _leq, ccd11, _prefetch)
19
20 #define _ccd11_sort_prefetch_extd(_ccd11, _carry, _pbuck, \
21     _iter_a, _iter_b, _leq) \
22     _cc_ll_sort_extd      (_ccd11, _carry, _pbuck, \
23     _iter_a, _iter_b, _leq, ccs11, _prefetch)
24
25
26
27 /* ccd11 iterators extended */
28
29
30 #define ccd11_iter_incr_prefetch(_iter, _parr, _pofs) \
31 ( \
32     *((_parr)[*(_pofs)]) = ((_iter)->curr.node), \
33     (_parr)[*(_pofs)] = &((_iter)->curr.node->PRV), \
34     *(_pofs) = (*(_pofs) + 1) % 64, \
35     __builtin_prefetch((_iter)->curr.node->PRV), \
36     ccs11_iter_at_tail((_iter)) ? (NULL) : \
37     ((_iter)->curr.node = (_iter)->curr.node->NXT)->NXT \
38 )
39
40
41
42 /* ccd11 integrity */
43
44
45 #define ccd11_is_sorted(_ccd11, _ptrue) \
46 \
47     ccd11_is_sorted_extd(_ccd11, DLEQ, _ptrue)
48

```

```

49 #define ccdll_is_sorted_extd(_ccdll, _leq, _ptrue) \
50 \
51         cc_ll_is_sorted_extd(_ccdll, _leq, _ptrue, ccdll)
52
53
54 #define ccdll_is_robust(_ccdll, _ptrue) \
55 \
56         cc_ll_is_robust(_ccdll, _ptrue, ccdll, CCDLL)
57
58
59
60 #endif

```

1.1.8 list/extd-ccsll.h

```
1  #ifndef OPENG3_LIST_EXTD_CCSLL_H
2  #define OPENG3_LIST_EXTD_CCSLL_H
3
4  #include "ccsll.h"
5  #include "extd-base.h"
6  #include "extd-ccd11.h"
7
8
9  /* ccsll operations extended */
10
11
12  #define ccsll_merge_prefetch(_ccsll_d, _ccsll_s) \
13 \
14      ccsll_merge_prefetch_extd(_ccsll_d, _ccsll_s, SLEQ_NEXT)
15
16  #define ccsll_merge_prefetch_extd(_ccsll_d, _ccsll_s, _leq) \
17 \
18      cc_ll_merge_extd      (_ccsll_d, _ccsll_s, _leq, ccsll, _prefetch)
19
20  #define _ccsll_merge_prefetch_extd(_iter_l, _iter_m, _iter_r, _leq) \
21 \
22  STATEMENT_ \
23  ( \
24      if (_unlikely((_iter_l)->cont == (_iter_m)->cont || \
25                  (_iter_l)->cont == (_iter_r)->cont || \
26                  (_iter_m)->cont != (_iter_r)->cont)) break; \
27 \
28      int _ofs = 0; \
29      __typeof__((_iter_l)->cont->pnode) _write, *_queue[64]; \
30 \
31      for (int _idx = 0; _idx < 64; _idx++) \
32          (_queue)[_idx] = &_write; \
33 \
34      ccsll_iter_head((_iter_l)); \
35      ccsll_iter_head((_iter_m)); \
36      ccsll_iter_head((_iter_r)); \
37 \
38      for (register int _len = 0; ; _len = 0) \
39      { \
40          while (!(ccsll_iter_at_end((_iter_l))) && _leq((_iter_l), (_iter_m))) \
41              ((void)ccd11_iter_incr_prefetch((_iter_l), _queue, &_ofs)); \
42 \
43          while (!(ccsll_iter_at_end((_iter_r))) && \
44                  (ccsll_iter_at_end((_iter_l)) || !_leq((_iter_l), (_iter_r)))) \
45              ((void)ccd11_iter_incr_prefetch((_iter_r), _queue, &_ofs, ++_len)); \
46 \
47          ccsll_move_range_extd((_iter_l), (_iter_m), (_iter_r), _len); \
48 \
```

```

49         if (ccsll_iter_at_end((_iter_m))) break; \
50 \
51         ccsll_iter_copy((_iter_l), (_iter_r)); \
52         ccsll_iter_init((_iter_r), (_iter_m)->cont) ; \
53         ccsll_iter_head((_iter_r)); \
54     } \
55 )
56
57
58
59 /* ccsll integrity */
60
61
62 #define ccsll_is_sorted(_ccsll, _ptrue) \
63 \
64         ccsll_is_sorted_extd(_ccsll, SLEQ, _ptrue)
65
66 #define ccsll_is_sorted_extd(_ccsll, _leq, _ptrue) \
67 \
68         cc_ll_is_sorted_extd(_ccsll, _leq, _ptrue, ccsll)
69
70
71 #define ccsll_is_robust(_ccsll, _ptrue) \
72 \
73         cc_ll_is_robust(_ccsll, _ptrue, ccsll, CCSLL)
74
75
76
77 #endif

```

1.2 base

1.2.1 base/mesg.h

```
1  #ifndef OPENG3_BASE_MESG_H
2  #define OPENG3_BASE_MESG_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  #include "misc.h"
8
9
10 /* error and fatal messages */
11
12
13 static const char CC_ERROR_MSG_MEMORY_LEAK[] = "Potential Memory Leak.";
14 static const char CC_ERROR_MSG_DOUBLE_FREE[] = "Potential Double Free.";
15
16 #define CC_ERROR(CC_ERROR_MSG) \
17 \
18 STATEMENT_ \
19 ( \
20     fprintf(stderr, "OpenGC^3::" "ERROR: %s\n", CC_ERROR_MSG); \
21 )
22
23
24 static const char CC_FATAL_MSG_MALLOC_FAIL[] = "Memory Allocation Failure.";
25 static const int  CC_FATAL_MSG_MALLOC_FAIL_EXITCODE = -1;
26
27 #define CC_FATAL(CC_FATAL_MSG) \
28 \
29 STATEMENT_ \
30 ( \
31     fprintf(stderr, "OpenGC^3::" "FATAL: %s\n", CC_FATAL_MSG); \
32     exit(CC_FATAL_MSG##_EXITCODE); \
33 )
34
35
36
37 #endif
```

1.2.2 base/misc.h

```
1  #ifndef OPENC3_BASE_MISC_H
2  #define OPENC3_BASE_MISC_H
3
4
5  /* syntax wrapper */
6
7  #define STATEMENT_(...)  do {__VA_ARGS__} while (0)
8  #define VOID_EXPR_(...)  ((__VA_ARGS__), ((void)0))
9
10
11 /* general macros */
12
13 #define MIN_2(A, B)  ((A) < (B) ? (A) : (B))
14 #define BITSOF(VAR)  (sizeof(VAR) * CHAR_BIT)
15 #define ELEM_OF(ARR)  (sizeof(ARR) / sizeof(ARR[0]))
16 #define UMAX_OF(VAR)  (~UINT64_C(0) >> (64 - BITSOF(VAR)))
17
18
19 /* compiler pragmas */
20
21 #define PRAGMA_NORMAL_BGN
22 #define PRAGMA_NORMAL_END
23
24 #define PRAGMA_PACKED_BGN  _Pragma("pack(push, 1)")
25 #define PRAGMA_PACKED_END  _Pragma("pack(pop)"      )
26
27
28 /* compiler extensions */
29
30 #ifndef CC_STRICT
31     #define _unlikely(_expr)  (__builtin_expect(!_expr, 0))
32     #define _prefetch(_addr)  (__builtin_prefetch(_addr))
33     #define _it_(_cont, _iter, _offset)  (&(_iter)[(_offset)])
34     #define _co_(_cont, _base, _offset)  ((_cont)->_co[(_base) + (_offset)])
35 #else
36     #define _unlikely(_expr)  (_expr)
37     #define _prefetch(_addr)  (_addr)
38     #define _it_(_cont, _base, _offset)  ((_cont)->_it[(_base) + (_offset)])
39     #define _co_(_cont, _base, _offset)  ((_cont)->_co[(_base) + (_offset)])
40 #endif // CC_STRICT
41
42
43 /* pointer layout */
44
45 #define XOR  lnk[0]
46 #define NXT  lnk[0]
47 #define PRV  lnk[1]
48
```



```

49 #define PRN lnk[0]
50 #define LFT lnk[1]
51 #define RGH lnk[2]
52
53
54 /* append line ID */
55
56 #define ADDID APPENDLINE
57 #define CONCATLINE(N, L) N ## _ ## L
58 #define EXPANDLINE(N, L) CONCATLINE(N, L)
59 #define APPENDLINE(NAME) EXPANDLINE(NAME, __LINE__)
60
61 #define CCDLL ADDID(CCDLL)
62 #define CCDLL_CONT ADDID(CCDLL_CONT)
63 #define CCDLL_NODE ADDID(CCDLL_NODE)
64 #define CCDLL_BLACK ADDID(CCDLL_BLACK)
65 #define CCDLL_ITER ADDID(CCDLL_ITER)
66 #define CCDLL_PTRS ADDID(CCDLL_PTRS)
67
68 #define CCSLL ADDID(CCSLL)
69 #define CCSLL_CONT ADDID(CCSLL_CONT)
70 #define CCSLL_NODE ADDID(CCSLL_NODE)
71 #define CCSLL_BLACK ADDID(CCSLL_BLACK)
72 #define CCSLL_ITER ADDID(CCSLL_ITER)
73 #define CCSLL_PTRS ADDID(CCSLL_PTRS)
74
75 #define CCXLL ADDID(CCXLL)
76 #define CCXLL_CONT ADDID(CCXLL_CONT)
77 #define CCXLL_NODE ADDID(CCXLL_NODE)
78 #define CCXLL_BLACK ADDID(CCXLL_BLACK)
79 #define CCXLL_ITER ADDID(CCXLL_ITER)
80 #define CCXLL_HDTL ADDID(CCXLL_HDTL)
81 #define CCXLL_PTRS ADDID(CCXLL_PTRS)
82
83 #define CCGBT ADDID(CCGBT)
84 #define CCGBT_CONT ADDID(CCGBT_CONT)
85 #define CCGBT_NODE ADDID(CCGBT_NODE)
86 #define CCGBT_BLACK ADDID(CCGBT_BLACK)
87 #define CCGBT_ITER ADDID(CCGBT_ITER)
88 #define CCGBT_PTRS ADDID(CCGBT_PTRS)
89
90 #define CCARR ADDID(CCARR)
91 #define CCARR_CONT ADDID(CCARR_CONT)
92
93
94
95 #endif

```

1.2.3 base/pool.h

```
1  #ifndef OPENG3_BASE_POOL_H
2  #define OPENG3_BASE_POOL_H
3
4  #include "mesg.h"
5  #include "misc.h"
6
7  #include <stdio.h>
8  #include <string.h>
9
10
11  /* safe allocation */
12
13
14  #define _safe_alloc(_void_ptr, _alloc_bytes) \
15 \
16  STATEMENT_ \
17  ( \
18      if ((_void_ptr) != NULL) \
19          CC_ERROR(CC_ERROR_MSG_MEMORY_LEAK); \
20 \
21      if (((_void_ptr) = malloc((_alloc_bytes))) == NULL) \
22          CC_FATAL(CC_FATAL_MSG_MALLOC_FAIL); \
23  )
24
25
26  #define _safe_free(_void_ptr) \
27 \
28  STATEMENT_ \
29  ( \
30      if ((_void_ptr) == NULL) \
31          CC_ERROR(CC_ERROR_MSG_DOUBLE_FREE); \
32 \
33      free((_void_ptr)); \
34      (_void_ptr) = NULL; \
35  )
36
37
38
39  /* cont / iter */
40
41
42  #define _cont_alloc(_cont)  _safe_alloc((_cont), sizeof(*(_cont)))
43
44  #define _cont_free(_cont)  _safe_free ((_cont))
45
46  #define _iter_alloc(_iter)  _safe_alloc((_iter), sizeof(*(_iter)))
47
48  #define _iter_free(_iter)  _safe_free ((_iter))
```

```

49
50
51
52  /* itarr */
53
54
55  #define _itarr_init(_cont, _ll_) \
56 \
57  STATEMENT_ \
58  ( \
59      _safe_alloc((_cont)->itarr, sizeof(*(_cont)->itarr)); \
60 \
61      for (int _idx = 0; _idx < (int)(ELEM_OF(*(_cont)->itarr)); _idx++) \
62          _ll_##_iter_init(&(*(_cont)->itarr][_idx]), (_cont)); \
63  )
64
65
66  #define _itarr_free(_cont) \
67 \
68  STATEMENT_ \
69  ( \
70      _safe_free((_cont)->itarr); \
71  )
72
73
74
75  /* _it / _co management */
76
77
78  #define _itco_total(_cont, _itco_) \
79 \
80      ((_cont)->_itco_##_base + (_cont)->_itco_##_limit)
81
82
83  #ifndef CC_STRICT
84
85  #define _it_init(_cont, _items, _iter, _ll_) \
86 \
87      __typeof__((*_cont)->_it) _iter[_items]; \
88 \
89      for (int _cnt = 0; _cnt < (_items); _cnt++) \
90          _ll_##_iter_init(_it((_cont), _iter, _cnt), (_cont))
91
92  #else
93
94  #define _it_init(_cont, _items, _base, _ll_) \
95 \
96      int _base; \
97 \
98      _itco_init((_cont), (_items), &(_base), _ll_##_iter_init, _it)

```

```

99
100 #endif // CC_STRICT
101
102
103 #define _co_init(_cont, _items, _base, _ll_) \
104 \
105     int _base; \
106 \
107     _itco_init((_cont), (_items), &(_base), _##_ll_##_init, _co)
108
109
110 #define _itco_init(_cont, _items, _pbase, _pinit, _itco_) \
111 \
112 STATEMENT_ \
113 ( \
114     _auxr_alloc((_cont), (_items), (_pbase), _itco_); \
115 \
116     for (int _idx = (*(_pbase)); _idx < (*(_pbase) + (_items)); _idx++) \
117         _pinit((_cont)->_itco_[_idx], (_cont), !((_cont)->_itco_[_idx])); \
118 )
119
120
121 #ifndef CC_STRICT
122
123 #define _it_clear(_cont, _items)
124
125 #else
126
127 #define _it_clear(_cont, _items) _itco_clear(_cont, _items, _it)
128
129 #endif // CC_STRICT
130
131
132 #define _co_clear(_cont, _items) \
133 \
134     _itco_clear(_cont, _items, _co)
135
136
137 #define _itco_clear(_cont, _items, _itco_) \
138 \
139     _auxr_clear(_cont, _items, _itco_)
140
141
142 #define _it_free(_cont) \
143 \
144 STATEMENT_ \
145 ( \
146     int _it_total = _itco_total((_cont), _it); \
147 \
148     for (int _idx_it = 0; _idx_it < _it_total; _idx_it++) \

```

```

149         _iter_free((_cont)->_it[_idx_it]);
150
151     _auxr_free((_cont), _it);
152 )
153
154
155 #define _co_free(_cont)
156
157 STATEMENT_
158 (
159     int _co_total = _itco_total((_cont), _co);
160
161     for (int _idx_co = 0; _idx_co < _co_total; _idx_co++)
162     {
163         _it_free    ((_cont)->_co[_idx_co]);
164         _block_free((_cont)->_co[_idx_co]);
165         _cont_free ((_cont)->_co[_idx_co]);
166     }
167
168     _auxr_free((_cont), _co);
169 )
170
171
172
173 /* auxr management */
174
175
176 #define _auxr_alloc(_cont, _items, _pbase, _itco_)
177
178 STATEMENT_
179 (
180     if ((_items) > (_cont)->_itco_##_limit && (_items) != 0)
181     {
182         size_t _size = sizeof(*(_cont)->_itco_);
183         int _ttl = _itco_total((_cont), _itco_);
184
185         void **_tmp = (void*)&((_cont)->_itco_);
186         *_tmp = realloc(*_tmp, (_size *
187                             (unsigned)((_cont)->_itco_##_base + (_items))));
188
189         memset((_cont)->_itco_ + _ttl, 0,
190                (_size * (unsigned)((_cont)->_itco_##_base + (_items) - _ttl)));
191
192         (_cont)->_itco_##_limit = (_items);
193     }
194
195     *(_pbase) = (_cont)->_itco_##_base;
196
197     (_cont)->_itco_##_base += (_items);
198     (_cont)->_itco_##_limit -= (_items);

```

```

199 )
200
201
202 #define _auxr_clear(_cont, _items, _itco_) \
203 \
204 STATEMENT_ \
205 ( \
206     (_cont)->_itco_##_base -= (_items); \
207     (_cont)->_itco_##_limit += (_items); \
208 )
209
210
211 #define _auxr_free(_cont, _itco_) \
212 \
213 STATEMENT_ \
214 ( \
215     if (_itco_total((_cont), _itco_) != 0) \
216         _safe_free ((_cont)->_itco_); \
217 \
218     (_cont)->_itco_##_base = 0; \
219     (_cont)->_itco_##_limit = 0; \
220 )
221
222
223
224 /* node / block management */
225
226
227 #define _node_alloc(_pnode, _cont) \
228 \
229 STATEMENT_ \
230 ( \
231     if ((_cont)->avsp == NULL) \
232     { \
233         if ((_cont)->vcnt == 0) \
234         { \
235             (_cont)->pblock = (_cont)->pool; \
236 \
237             if ((_cont)->pool != NULL && (_cont)->last != 0) \
238                 (_cont)->pool = (_cont)->pool->bnxt; \
239 \
240             if ((_cont)->last == 0) \
241                 (_cont)->vcnt = ((_cont)->last = (_cont)->start); \
242             else \
243                 (_cont)->vcnt = ((_cont)->last < (_cont)->thrsh) ? \
244                     ((_cont)->last *= (_cont)->ratio) : \
245                     ((_cont)->last = (_cont)->thrsh); \
246 \
247             if ((_cont)->pool == NULL) \
248             { \

```

```

249         _safe_alloc((_cont)->pool, (sizeof(*(_cont)->pblock)) + \
250                                (sizeof(*(_cont)->pblock->nodes)) * \
251                                (size_t)((_cont)->vcnt)); \
252 \
253         (_cont)->pool->bprv = (_cont)->pblock; \
254         (_cont)->pool->ncnt = (_cont)->vcnt; \
255 \
256         (_cont)->pool->bnxt = NULL; \
257         if ((_cont)->pool->bprv != NULL) \
258             (_cont)->pool->bprv->bnxt = (_cont)->pool; \
259     } \
260 } \
261 \
262     (_pnode) = &((_cont)->pool->nodes[--(_cont)->vcnt]); \
263 } \
264 else \
265 { \
266     (_pnode) = (_cont)->avsp; \
267     (_cont)->avsp = (_cont)->avsp->lnk[0]; \
268 } \
269 ) \
270 \
271 \
272 #define _node_clear(_pnode, _cont) \
273 \
274 STATEMENT_ \
275 ( \
276     (_pnode)->lnk[0] = (_cont)->avsp; \
277     (_cont)->avsp = (_pnode); \
278 ) \
279 \
280 \
281 #define _block_free(_cont) \
282 \
283 STATEMENT_ \
284 ( \
285     while ((_cont)->pool != NULL) \
286     { \
287         (_cont)->pblock = (_cont)->pool; \
288         (_cont)->pool = (_cont)->pool->bprv; \
289         _safe_free((_cont)->pblock); \
290     } \
291 ) \
292 \
293 \
294 \
295 #endif

```

1.2.4 base/snym.h

```
1 #ifndef OPENG33_BASE_SNYM_H
2 #define OPENG33_BASE_SNYM_H
3
4
5 /* dereference */
6
7 // ccdll
8 #define DREF(_iter)          ((_iter)->curr.node->val)
9 #define DREF_PREV(_iter)    ((_iter)->curr.node->PRV->val)
10 #define DREF_NEXT(_iter)    ((_iter)->curr.node->NXT->val)
11
12 // ccsl1
13 #define SREF(_iter)          ((_iter)->curr.node->val)
14 #define SREF_NEXT(_iter)    ((_iter)->curr.node->NXT->val)
15
16 // ccxll
17 #define XREF(_iter)          ((_iter)->curr.node->val)
18 #define XREF_PREV(_iter)    ((_iter)->prev.node->val)
19 #define XREF_NEXT(_iter)    ((_iter)->next.node->val)
20
21 // cc[dsx]ll
22 #define LREF(_iter)          ((_iter)->curr.node->val)
23
24 // ccgbt
25 #define GREF(_iter)          ((_iter)->curr.node->val)
26 #define GREF_PARENT(_iter)  ((_iter)->curr.node->PRN->val)
27 #define GREF_LEFT(_iter)    ((_iter)->curr.node->LFT->val)
28 #define GREF_RIGHT(_iter)   ((_iter)->curr.node->RGH->val)
29
30
31 /* abbreviation */
32
33 // ccdll
34 #define DLEQ          ccdll_comp_leq
35 #define DLEQ_PREV    ccdll_comp_leq_prev
36 #define DLEQ_NEXT    ccdll_comp_leq_next
37 #define DGEQ          ccdll_comp_geq
38 #define DGEQ_PREV    ccdll_comp_geq_prev
39 #define DGEQ_NEXT    ccdll_comp_geq_next
40
41 // ccsl1
42 #define SLEQ          ccsl1_comp_leq
43 #define SLEQ_NEXT    ccsl1_comp_leq_next
44 #define SGEQ          ccsl1_comp_geq
45 #define SGEQ_NEXT    ccsl1_comp_geq_next
46
47 // ccxll
48 #define XLEQ          ccxll_comp_leq
```



```

49 #define XLEQ_PREV    ccxll_comp_leq_prev
50 #define XLEQ_NEXT    ccxll_comp_leq_next
51 #define XGEQ          ccxll_comp_geq
52 #define XGEQ_PREV    ccxll_comp_geq_prev
53 #define XGEQ_NEXT    ccxll_comp_geq_next
54
55 // cc[dsx]ll
56 #define ITER(_cont)          (ITER_NTH(_cont, 0))
57 #define ITER_NTH(_cont, _nth_it) (&(*(_cont)->itarr)[(_nth_it)])
58
59 // ccarr
60 #define ELEM(_cont)          (ELEM_NTH(_cont, 0))
61 #define ELEM_NTH(_cont, _nth_el) ((_cont).arr[(_nth_el)])
62
63
64 #endif

```