The Core Implementations of The OpenGC^3 Project

Dong Nai-Jia

November 13, 2017

Abstract

OpenGC^3 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

1.1 list

1.1.1 list/base.h

```
#ifndef OPENGC3_LIST_BASE_H
#define OPENGC3_LIST_BASE_H


/* exclusive or */


#define XOR2(_addr_a, _addr_b)                                          \
(                                                                       \
    (void*)((uintptr_t)(void*)(_addr_a) ^                               \
            (uintptr_t)(void*)(_addr_b))                                \
)


#define XOR3(_addr_a, _addr_b, _addr_c)                                 \
(                                                                       \
    (void*)((uintptr_t)(void*)(_addr_a) ^                               \
            (uintptr_t)(void*)(_addr_b) ^                               \
            (uintptr_t)(void*)(_addr_c))                                \
)


#define XOR2_SWAP(_addr_a, _addr_b)                                     \
                                                                        \
VOID_EXPR_                                                              \
(                                                                       \
    (_addr_a) = XOR2((_addr_a), (_addr_b)),                             \
    (_addr_b) = XOR2((_addr_a), (_addr_b)),                             \
    (_addr_a) = XOR2((_addr_a), (_addr_b))                              \
)



/* cc_ll initialize */


#define cc_ll_init_extd(_cc_ll, _start, _ratio, _thrsh, _ll_)           \
                                                                        \
STATEMENT_                                                              \
(                                                                       \
    (_cc_ll) = NULL;                                                    \
                                                                        \
    _##_ll_##_init_extd((_cc_ll), (_start), (_ratio), (_thrsh), 1);     \
                                                                        \
    _itarr_init((_cc_ll), _ll_);                                        \
```

```
    _##_ll_##_iter_init((_cc_ll)->_iter, (_cc_ll), 1);                          \
)


#define _cc_ll_init_extd(_cc_ll, _start, _ratio, _thrsh, _alloc, _ll_)         \
                                                                               \
STATEMENT_                                                                     \
(                                                                              \
    if ((_alloc))  _cont_alloc((_cc_ll));                                      \
                                                                               \
    _##_ll_##_init_core((_cc_ll));                                             \
    _##_ll_##_init_info((_cc_ll), (_start), (_ratio), (_thrsh));               \
)


#define _cc_ll_init_core(_cc_ll, _ll_)                                         \
                                                                               \
VOID_EXPR_                                                                     \
(                                                                              \
    _##_ll_##_init_seed((_cc_ll)),                                             \
                                                                               \
    (_cc_ll)->avsp = (_cc_ll)->pnode  = NULL,                                  \
    (_cc_ll)->pool = (_cc_ll)->pblock = NULL,                                  \
                                                                               \
    (_cc_ll)->itarr = NULL,                                                    \
    (_cc_ll)->_iter = NULL,                                                    \
    (_cc_ll)->_it   = NULL,                                                    \
    (_cc_ll)->_co   = NULL,                                                    \
    (_cc_ll)->_it_base = (_cc_ll)->_it_limit = 0,                              \
    (_cc_ll)->_co_base = (_cc_ll)->_co_limit = 0                               \
)


#define _cc_ll_init_info(_cc_ll, _start, _ratio, _thrsh)                       \
                                                                               \
VOID_EXPR_                                                                     \
(                                                                              \
    (_cc_ll)->start = ((_start) > 0) ? (_start) : 1,                           \
    (_cc_ll)->ratio = ((_ratio) > 0) ? (_ratio) : 1,                           \
    (_cc_ll)->thrsh = ((_thrsh) > (_cc_ll)->start) ? (_thrsh) : (_cc_ll)->start\
)


#define _cc_ll_iter_init(_iter, _cc_ll, _alloc, _ll_)                          \
                                                                               \
STATEMENT_                                                                     \
(                                                                              \
    if ((_alloc))  _iter_alloc((_iter));                                       \
                                                                               \
    _ll_##_iter_init((_iter), (_cc_ll));                                       \
```

```
)


/* cc_ll destroy */


#define cc_ll_free(_cc_ll)                                                  \
                                                                            \
STATEMENT_                                                                  \
(                                                                           \
    _it_free((_cc_ll));                                                     \
    _co_free((_cc_ll));                                                     \
                                                                            \
    _iter_free ((_cc_ll)->_iter);                                           \
    _itarr_free((_cc_ll));                                                  \
    _block_free((_cc_ll));                                                  \
    _cont_free ((_cc_ll));                                                  \
)


/* cc_ll operations */


#define cc_ll_merge_extd(_cc_ll_d, _cc_ll_s, _leq, _ll_, _opt_)             \
                                                                            \
STATEMENT_                                                                  \
(                                                                           \
     if (_unlikely(_ll_##_empty((_cc_ll_s))))  break;                       \
                                                                            \
    _it_init((_cc_ll_d), 1, _base_m1, _ll_);                                \
    _it_init((_cc_ll_s), 2, _base_m2, _ll_);                                \
                                                                            \
    _##_ll_##_merge##_opt_##_extd(_it_((_cc_ll_d), _base_m1, 0),            \
                                  _it_((_cc_ll_s), _base_m2, 0),            \
                                  _it_((_cc_ll_s), _base_m2, 1), _leq);     \
    _it_clear((_cc_ll_d), 1);                                               \
    _it_clear((_cc_ll_s), 2);                                               \
)


#define _cc_ll_merge_extd(_iter_l, _iter_m, _iter_r, _leq, _ll_, _opt_)     \
                                                                            \
STATEMENT_                                                                  \
(                                                                           \
    _ll_##_iter_tail ((_iter_l));                                           \
    _ll_##_iter_begin((_iter_m));                                           \
    _ll_##_iter_tail ((_iter_r));                                           \
                                                                            \
```

```
    _ll_##_move_range_extd( (_iter_l), (_iter_m), (_iter_r),                    \
                            _ll_##_size((_iter_m)->cont));                      \
                                                                                \
    _ll_##_iter_begin((_iter_l));                                               \
    _ll_##_iter_init ((_iter_r), (_iter_l)->cont);                              \
    _ll_##_iter_tail ((_iter_r));                                               \
                                                                                \
    _ll_##_merge_range##_opt_##_extd((_iter_l), (_iter_m), (_iter_r), _leq);    \
)


#define cc_ll_merge_range_extd(_iter_l, _iter_m, _iter_r, _leq, _ll_, _opt_)    \
                                                                                \
STATEMENT_                                                                      \
(                                                                               \
    _it_init((_iter_l)->cont, 1, _base_m3, _ll_);                               \
                                                                                \
    _##_ll_##_merge_range##_opt_##_extd((_iter_l), (_iter_m), (_iter_r),        \
                                    _it_((_iter_l)->cont, _base_m3, 0), _leq);  \
                                                                                \
    _it_clear((_iter_l)->cont, 1);                                              \
)


#define _cc_ll_merge_range_extd(_iter_l, _iter_m, _iter_r, _iter_x, _leq, _ll_)\
                                                                                \
STATEMENT_                                                                      \
(                                                                               \
    if (_unlikely((_iter_l)->cont != (_iter_m)->cont ||                         \
                  (_iter_m)->cont != (_iter_r)->cont))  break;                  \
                                                                                \
    _ll_##_iter_copy((_iter_x), (_iter_m));                                     \
                                                                                \
    for (register int _neq; ; )                                                 \
    {                                                                           \
        while ((_neq = ((_iter_l)->curr.node                                    \
                    != (_iter_m)->curr.node)) &&  _leq((_iter_l), (_iter_m))) \
            (void)(_ll_##_iter_incr((_iter_l)));                                \
                                                                                \
        if (!(_neq))                                                            \
        {                                                                       \
            _ll_##_iter_copy((_iter_l), (_iter_r));                             \
            _ll_##_iter_copy((_iter_m), (_iter_r));  break;                     \
        }                                                                       \
                                                                                \
        (void)(_ll_##_iter_incr((_iter_x)));                                    \
                                                                                \
        while ((_neq = ((_iter_x)->curr.node                                    \
                    != (_iter_r)->curr.node)) && !_leq((_iter_l), (_iter_x))) \
            (void)(_ll_##_iter_incr((_iter_x)));                                \
```

```
                                                                        \
        _ll_##_move_range((_iter_l), (_iter_m), (_iter_x));             \
        _ll_##_iter_copy ((_iter_m), (_iter_x));                        \
                                                                        \
        if (!(_neq))                                                    \
        {                                                               \
            _ll_##_iter_copy((_iter_l), (_iter_x));                     \
            _ll_##_iter_copy((_iter_r), (_iter_x));  break;             \
        }                                                               \
    }                                                                   \
)


#define cc_ll_sort_extd(_cc_ll, _leq, _ll_, _opt_)                      \
                                                                        \
STATEMENT_                                                              \
(                                                                       \
    if (_unlikely(_ll_##_size((_cc_ll)) <= 1))  break;                  \
                                                                        \
    int _buck = (int)(log2(_ll_##_size((_cc_ll)))) + 1;                 \
                                                                        \
    _co_init((_cc_ll), 1 + _buck, _base_s1, _ll_);                      \
    _it_init((_cc_ll), 2        , _base_s2, _ll_);                      \
                                                                        \
    _##_ll_##_sort##_opt_##_extd(       (_cc_ll),                       \
                                  _co_((_cc_ll), _base_s1, 0),          \
                                 &(_co_((_cc_ll), _base_s1, 1)),        \
                                  _it_((_cc_ll), _base_s2, 0),          \
                                  _it_((_cc_ll), _base_s2, 1), _leq);   \
                                                                        \
    _co_clear((_cc_ll), _buck + 1);                                     \
    _it_clear((_cc_ll),  2);                                            \
)


#define _cc_ll_sort_extd(_cc_ll,  _carry,  _pcont,                      \
                         _iter_a, _iter_b, _leq, _ll_, _opt_)           \
STATEMENT_                                                              \
(                                                                       \
    int _fill = 0, _curr;                                               \
                                                                        \
    do                                                                  \
    {   _ll_##_iter_init ((_iter_a), (_carry));                         \
        _ll_##_iter_init ((_iter_b), (_cc_ll));                         \
        _ll_##_move_begin((_iter_a), (_iter_b));                        \
                                                                        \
        for (_curr = 0; _curr != _fill &&                               \
                        !(_ll_##_empty((_pcont)[_curr])); _curr++)      \
        {                                                               \
            _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
260 /* cc_ll iterators */
261
262
263 #define cc_ll_iter_distance(_iter_a, _iter_b, _pdist, _ll_ /* != ccsll */ )   \
264                                                                              \
265 STATEMENT_                                                                   \
266 (                                                                            \
267     _it_init((_iter_a)->cont, 1, _base_d1, _ll_);                            \
268                                                                              \
269     _ll_##_iter_copy(_it_((_iter_a)->cont, _base_d1, 0), (_iter_a));         \
270                                                                              \
271     STATEMENT_                                                               \
272     (                                                                        \
273         (*(_pdist)) = 0;                                                     \
274         if ((_iter_a)->cont != (_iter_b)->cont)  break;                      \
275                                                                              \
276         while ((_iter_a)->curr.node != (_iter_b)->curr.node && ++(*(_pdist)))  \
277                if (!(_ll_##_iter_incr((_iter_a))))  break;                   \
278                                                                              \
279         if ((_iter_a)->curr.node == (_iter_b)->curr.node)  break;            \
280         else  (*(_pdist)) = 0;                                               \
281                                                                              \
282         _ll_##_iter_copy((_iter_a), _it_((_iter_a)->cont, _base_d1, 0));     \
283                                                                              \
284         while ((_iter_a)->curr.node != (_iter_b)->curr.node && --(*(_pdist)))  \
285                if (!(_ll_##_iter_decr((_iter_a))))  break;                   \
286                                                                              \
287         if ((_iter_a)->curr.node == (_iter_b)->curr.node)  break;            \
288         else  (*(_pdist)) = 0;                                               \
289     );                                                                       \
290                                                                              \
291     _ll_##_iter_copy((_iter_a), _it_((_iter_a)->cont, _base_d1, 0));         \
292                                                                              \
293     _it_clear((_iter_a)->cont, 1);                                           \
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

```
#ifndef OPENGC3_LIST_CCXLL_H
#define OPENGC3_LIST_CCXLL_H

#include "base.h"
#include "../base/pool.h"
#include "../base/misc.h"
#include "../base/snym.h"

#include <math.h>
#include <stddef.h>
#include <stdint.h>
#include <string.h>


/* ccxll create */


#define ccxll(elem_t)                                                   \
                                                                        \
        ccxll_extd(elem_t, 1, NORMAL)

#define ccxll_pckd(elem_t)                                              \
                                                                        \
        ccxll_extd(elem_t, 1, PACKED)

#define ccxll_extd(elem_t, _n_iter, _ALIGN_)                            \
                                                                        \
        typedef ccxll_struct_extd(elem_t, _n_iter, _ALIGN_) *CCXLL;  CCXLL

#define ccxll_type(elem_t)                                              \
                                                                        \
        typedef ccxll_struct_extd(elem_t, 1, NORMAL) *


#define link_t void*

#define ccxll_struct(elem_t)                                            \
                                                                        \
        ccxll_struct_extd(elem_t, 1, NORMAL)

#define ccxll_struct_pckd(elem_t)                                       \
                                                                        \
        ccxll_struct_extd(elem_t, 1, PACKED)

#define ccxll_struct_extd(elem_t, _n_iter, _ALIGN_)                     \
                                                                        \
    struct CCXLL_CONT                                                   \
    {                                                                   \
```

```
        int size,  last,  vcnt;                 /* size and node record */   \
        int start, ratio, thrsh;                /* block increment info */   \
                                                                             \
        struct CCXLL_NODE                                                    \
        {   link_t lnk[1];                                                   \
            elem_t val;                         /* val with an xor link */   \
        }   *avsp, *pnode, swap;                /* available space list */   \
                                                                             \
        union  CCXLL_HDTL                                                    \
        {   link_t lnk[1];                                                   \
            struct CCXLL_NODE *stnl;            /* points to same addr. */   \
        }   head, tail;                         /* two pseudo sentinels */   \
                                                                             \
        struct CCXLL_BLCK                                                    \
        {   struct CCXLL_BLCK *bprv, *bnxt;     /* next and prev blocks */   \
            PRAGMA_##_ALIGN_##_BGN              /* packed pragma starts */   \
            int ncnt;                           /* the item of the node */   \
            struct CCXLL_NODE nodes[];          /* node structure array */   \
            PRAGMA_##_ALIGN_##_END              /* the pragma ends here */   \
        }   *pool, *pblock;                     /* points to 1-st block */   \
                                                                             \
        struct CCXLL_ITER                                                    \
        {   union CCXLL_PTRS                                                 \
            {   link_t lnk[1];                                               \
                struct CCXLL_NODE *node;                                     \
            }   prev, curr, next;               /* points to p/v/n node */   \
            struct CCXLL_CONT *cont;            /* points to ccxll body */   \
        }   (*itarr)[_n_iter], *_iter, **_it;                                \
                                                                             \
        struct CCXLL_CONT **_co;                /* internal use _it _co */   \
                                                                             \
        int _it_base, _it_limit;                                             \
        int _co_base, _co_limit;                                             \
    }


/* ccxll initialize */


#define ccxll_init(_ccxll)                                                   \
                                                                             \
        ccxll_init_extd(_ccxll,     16,      2,  65536)

#define ccxll_init_extd(_ccxll, _start, _ratio, _thrsh)                      \
                                                                             \
        cc_ll_init_extd(_ccxll, _start, _ratio, _thrsh, ccxll)


#define _ccxll_init(_ccxll_dst, _ccxll_src, _alloc)                          \
```

```
                                                                        \
        _ccxll_init_extd(_ccxll_dst, -1,     -1,     -1, _alloc)

#define _ccxll_init_extd(_ccxll, _start, _ratio, _thrsh, _alloc)        \
                                                                        \
        _cc_ll_init_extd(_ccxll, _start, _ratio, _thrsh, _alloc, ccxll)


#define _ccxll_init_core(_ccxll)                                        \
                                                                        \
        _cc_ll_init_core(_ccxll, ccxll)


#define _ccxll_init_seed(_ccxll)                                        \
                                                                        \
VOID_EXPR_                                                              \
(                                                                       \
    (_ccxll)->size = 0,                                                 \
    (_ccxll)->last = (_ccxll)->vcnt = 0,                                \
                                                                        \
    (_ccxll)->head.XOR = &((_ccxll)->tail),                             \
    (_ccxll)->tail.XOR = &((_ccxll)->head)                              \
)


#define _ccxll_init_info(_ccxll, _start, _ratio, _thrsh)                \
                                                                        \
        _cc_ll_init_info(_ccxll, _start, _ratio, _thrsh)


#define ccxll_iter_init(_iter, _ccxll)                                  \
                                                                        \
VOID_EXPR_                                                              \
(                                                                       \
    (_iter)->prev.XOR = NULL,                                           \
    (_iter)->curr.XOR = NULL,                                           \
    (_iter)->next.XOR = NULL,                                           \
    (_iter)->cont = (_ccxll)                                            \
)


#define _ccxll_iter_init(_iter, _ccxll, _alloc)                         \
                                                                        \
        _cc_ll_iter_init(_iter, _ccxll, _alloc, ccxll)



/* ccxll destroy */


```

```
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                                                                                \
```

```
    (_ccxll)->pnode->XOR       = XOR2(&((_ccxll)->_hdtl_.XOR),                \
                                      (_ccxll)->_hdtl_.XOR);                  \
                                                                              \
    (_ccxll)->_hdtl_.stnl->XOR = XOR3(&((_ccxll)->_hdtl_.XOR),                \
                                      (_ccxll)->_hdtl_.stnl->XOR,             \
                                     &((_ccxll)->pnode->XOR));                \
                                                                              \
    (_ccxll)->_hdtl_.XOR = &((_ccxll)->pnode->XOR);                           \
                                                                              \
    (_ccxll)->size++;                                                         \
)


#define  ccxll_pop_front(_ccxll)  _ccxll_pop(_ccxll, head)

#define  ccxll_pop_back(_ccxll)   _ccxll_pop(_ccxll, tail)

#define _ccxll_pop(_ccxll, _hdtl_)                                            \
                                                                              \
STATEMENT_                                                                    \
(                                                                             \
    if (ccxll_empty((_ccxll)))  break;                                        \
                                                                              \
    (_ccxll)->pnode = (_ccxll)->_hdtl_.stnl;                                  \
                                                                              \
    (_ccxll)->_hdtl_.XOR       = XOR2(&((_ccxll)->_hdtl_.XOR),                \
                                      (_ccxll)->_hdtl_.stnl->XOR);            \
                                                                              \
    (_ccxll)->_hdtl_.stnl->XOR = XOR3(&((_ccxll)->_hdtl_.XOR),                \
                                      (_ccxll)->_hdtl_.stnl->XOR,             \
                                     &((_ccxll)->pnode->XOR));                \
                                                                              \
    _node_clear((_ccxll)->pnode, (_ccxll));                                   \
                                                                              \
    (_ccxll)->size--;                                                         \
)


#define ccxll_insert(_iter, _val)                                             \
                                                                              \
STATEMENT_                                                                    \
(                                                                             \
    if (ccxll_iter_at_head((_iter)))  break;                                  \
                                                                              \
    _node_alloc((_iter)->cont->pnode, (_iter)->cont);                         \
                                                                              \
    (_iter)->cont->pnode->val = (_val);                                       \
                                                                              \
    (_iter)->next.node = (_iter)->curr.node;                                  \
    (_iter)->curr.node = (_iter)->cont->pnode;                                \
```

```
                                                                                \
    (_iter)->curr.node->XOR = XOR2((_iter)->prev.XOR, (_iter)->next.XOR);       \
                                                                                \
    (_iter)->prev.node->XOR = XOR3((_iter)->prev.node->XOR,                     \
                                   (_iter)->next.XOR,                           \
                                  &((_iter)->cont->pnode->XOR));                \
    (_iter)->next.node->XOR = XOR3((_iter)->next.node->XOR,                     \
                                   (_iter)->prev.XOR,                           \
                                  &((_iter)->cont->pnode->XOR));                \
    (_iter)->cont->size++;                                                      \
)


#define ccxll_erase(_iter)                                                      \
                                                                                \
STATEMENT_                                                                      \
(                                                                               \
    if (ccxll_iter_at_head((_iter)) || ccxll_iter_at_tail((_iter)))  break;     \
                                                                                \
    (_iter)->prev.node->XOR = XOR3((_iter)->prev.node->XOR,                     \
                                   (_iter)->next.XOR, (_iter)->curr.XOR);       \
    (_iter)->next.node->XOR = XOR3((_iter)->next.node->XOR,                     \
                                   (_iter)->prev.XOR, (_iter)->curr.XOR);       \
                                                                                \
    _node_clear((_iter)->curr.node, (_iter)->cont);                             \
                                                                                \
    (_iter)->curr.XOR =      (_iter)->next.XOR;                                 \
    (_iter)->next.XOR = XOR2((_iter)->curr.node->XOR, (_iter)->prev.XOR);       \
                                                                                \
    (_iter)->cont->size--;                                                      \
)


#define ccxll_swap(_ccxll_a, _ccxll_b)                                          \
                                                                                \
STATEMENT_                                                                      \
(                                                                               \
    XOR2_SWAP((_ccxll_a), (_ccxll_b));                                          \
)


#define ccxll_resize(_ccxll, _items, _val)                                      \
                                                                                \
STATEMENT_                                                                      \
(                                                                               \
    int _size = ccxll_size((_ccxll)) - (_items);                                \
                                                                                \
         if (_size > 0)  while(_size--)  ccxll_pop_back ((_ccxll));             \
    else if (_size < 0)  while(_size++)  ccxll_push_back((_ccxll), (_val));     \
)
```

```
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                                                                                \
```

```
    (_iter_p)->prev.XOR =      (_iter_i)->curr.XOR;                             \
    (_iter_p)->next.XOR = XOR2((_iter_p)->prev.XOR, (_iter_p)->curr.node->XOR);\
)


#define ccxll_move_begin(_iter_a, _iter_b)                                      \
                                                                                \
STATEMENT_                                                                      \
(                                                                               \
    ccxll_iter_begin((_iter_a));                                                \
    ccxll_iter_begin((_iter_b));                                                \
                                                                                \
    ccxll_move((_iter_a), (_iter_b));                                           \
)


#define ccxll_move_range(_iter_p, _iter_l, _iter_r)                             \
                                                                                \
        ccxll_move_range_extd(_iter_p, _iter_l, _iter_r,    -1)

#define ccxll_move_range_extd(_iter_p, _iter_l, _iter_r, _dist)  /* [l, r) */   \
                                                                                \
STATEMENT_                                                                      \
(                                                                               \
    if (_unlikely(ccxll_iter_at_head ((_iter_p)) ||                             \
                  ccxll_iter_at_head ((_iter_l))))  break;                      \
                                                                                \
    if (_unlikely((_iter_l)->curr.node == (_iter_r)->curr.node))  break;        \
                                                                                \
    if (_unlikely((_iter_l)->cont != (_iter_r)->cont))  break;                  \
                                                                                \
    if (_unlikely((_iter_p)->cont != (_iter_l)->cont))                          \
    {                                                                           \
        int _dist_m = (_dist);                                                  \
                                                                                \
        if (_dist_m <  0)                                                       \
            ccxll_iter_distance((_iter_l), (_iter_r), &_dist_m);                \
                                                                                \
        if (_dist_m <= 0)  break;                                               \
                                                                                \
        (_iter_p)->cont->size += _dist_m;                                       \
        (_iter_l)->cont->size -= _dist_m;                                       \
        (_iter_l)->cont = (_iter_p)->cont;                                      \
    }                                                                           \
                                                                                \
    link_t _p_c = (_iter_p)->curr.XOR;                                          \
    link_t _l_c = (_iter_l)->curr.XOR;                                          \
    link_t _r_c = (_iter_r)->curr.XOR;                                          \
    link_t _p_p = (_iter_p)->prev.XOR;                                          \
    link_t _l_p = (_iter_l)->prev.XOR;                                          \
```

```
    link_t _r_p = (_iter_r)->prev.XOR;                                          \
                                                                                \
    (_iter_p)->prev.node->XOR = XOR3((_iter_p)->prev.node->XOR, _p_c, _l_c);    \
    (_iter_l)->prev.node->XOR = XOR3((_iter_l)->prev.node->XOR, _l_c, _r_c);    \
    (_iter_r)->prev.node->XOR = XOR3((_iter_r)->prev.node->XOR, _r_c, _p_c);    \
                                                                                \
    (_iter_p)->curr.node->XOR = XOR3((_iter_p)->curr.node->XOR, _p_p, _r_p);    \
    (_iter_l)->curr.node->XOR = XOR3((_iter_l)->curr.node->XOR, _l_p, _p_p);    \
    (_iter_r)->curr.node->XOR = XOR3((_iter_r)->curr.node->XOR, _r_p, _l_p);    \
                                                                                \
    if (_unlikely((_iter_p)->next.XOR == _l_c))  (_iter_p)->next.XOR = _r_c;    \
    if           ((_iter_l)->next.XOR == _r_c)   (_iter_l)->next.XOR = _p_c;    \
    if (_unlikely((_iter_r)->next.XOR == _p_c))  (_iter_r)->next.XOR = _l_c;    \
                                                                                \
    (_iter_p)->prev.XOR = XOR2((_iter_p)->curr.node->XOR, (_iter_p)->next.XOR);\
    (_iter_l)->prev.XOR = XOR2((_iter_l)->curr.node->XOR, (_iter_l)->next.XOR);\
    (_iter_r)->prev.XOR = XOR2((_iter_r)->curr.node->XOR, (_iter_r)->next.XOR);\
)


#define  ccxll_merge(_ccxll_d, _ccxll_s)                                        \
                                                                                \
         ccxll_merge_extd(_ccxll_d, _ccxll_s, XLEQ)

#define  ccxll_merge_extd(_ccxll_d, _ccxll_s, _leq)                             \
                                                                                \
         cc_ll_merge_extd(_ccxll_d, _ccxll_s, _leq, ccxll, )

#define _ccxll_merge_extd(_iter_l, _iter_m, _iter_r, _leq)                      \
                                                                                \
        _cc_ll_merge_extd(_iter_l, _iter_m, _iter_r, _leq, ccxll, )


#define  ccxll_merge_range(_iter_l, _iter_m, _iter_r)                           \
                                                                                \
         ccxll_merge_range_extd(_iter_l, _iter_m, _iter_r, XLEQ)

#define  ccxll_merge_range_extd(_iter_l, _iter_m, _iter_r, _leq)                \
                                                                                \
         cc_ll_merge_range_extd(_iter_l, _iter_m, _iter_r, _leq, ccxll, )

#define _ccxll_merge_range_extd(_iter_l, _iter_m, _iter_r, _iter_x, _leq)       \
                                                                                \
        _cc_ll_merge_range_extd(_iter_l, _iter_m, _iter_r, _iter_x, _leq, ccxll)


#define  ccxll_sort(_ccxll)                                                     \
                                                                                \
         ccxll_sort_extd(_ccxll, XLEQ)

```

```
#define  ccxll_sort_extd(_ccxll, _leq)                                          \
                                                                                \
         cc_ll_sort_extd(_ccxll, _leq, ccxll, )

#define _ccxll_sort_extd(_ccxll, _carry, _pbuck, _iter_a, _iter_b, _leq)       \
                                                                                \
        _cc_ll_sort_extd(_ccxll, _carry, _pbuck, _iter_a, _iter_b, _leq, ccxll,)


#define ccxll_reverse_range(_iter_l, _iter_r)                                  \
                                                                               \
        ccxll_reverse_range_extd(_iter_l, _iter_r,  0)                         \

#define ccxll_reverse_range_extd(_iter_l, _iter_r, _flag_swap_iters)           \
                                                                               \
STATEMENT_                                                                     \
(                                                                              \
    if (_unlikely((_iter_l)->cont != (_iter_r)->cont))  break;                 \
                                                                               \
    if (_unlikely((_iter_l)->curr.XOR == (_iter_r)->curr.XOR))  break;         \
                                                                               \
    link_t _x_in = XOR2((_iter_l)->curr.XOR, (_iter_r)->curr.XOR);             \
    link_t _x_ex = XOR2((_iter_l)->prev.XOR, (_iter_r)->next.XOR);             \
                                                                               \
    (_iter_l)->prev.node->XOR = XOR2((_iter_l)->prev.node->XOR, _x_in);        \
    (_iter_r)->next.node->XOR = XOR2((_iter_r)->next.node->XOR, _x_in);        \
                                                                               \
    (_iter_l)->curr.node->XOR = XOR2((_iter_l)->curr.node->XOR, _x_ex);        \
    (_iter_r)->curr.node->XOR = XOR2((_iter_r)->curr.node->XOR, _x_ex);        \
                                                                               \
    switch ((_flag_swap_iters))                                                \
    {                                                                          \
        case 0:                                                                \
        XOR2_SWAP((_iter_l)->prev.XOR, (_iter_l)->next.XOR);                   \
        XOR2_SWAP((_iter_r)->prev.XOR, (_iter_r)->next.XOR);                   \
        (_iter_l)->next.XOR = XOR2((_iter_l)->next.XOR, _x_ex);                \
        (_iter_r)->prev.XOR = XOR2((_iter_r)->prev.XOR, _x_ex);                \
        break;                                                                 \
                                                                               \
        case 1:  default:                                                      \
        XOR2_SWAP((_iter_l)->curr.XOR, (_iter_r)->curr.XOR);                   \
        XOR2_SWAP((_iter_l)->next.XOR, (_iter_r)->prev.XOR);                   \
    }                                                                          \
)


/* ccxll comparator */


```

```
#define ccxll_comp_leq(_iter_a, _iter_b)       (XREF((_iter_a)) <=            \
                                                XREF((_iter_b)))

#define ccxll_comp_leq_prev(_iter_a, _iter_b)  (XREF_PREV((_iter_a)) <=       \
                                                XREF_PREV((_iter_b)))

#define ccxll_comp_leq_next(_iter_a, _iter_b)  (XREF_NEXT((_iter_a)) <=       \
                                                XREF_NEXT((_iter_b)))

#define ccxll_comp_geq(_iter_a, _iter_b)       (XREF((_iter_a)) >=            \
                                                XREF((_iter_b)))

#define ccxll_comp_geq_prev(_iter_a, _iter_b)  (XREF_PREV((_iter_a)) >=       \
                                                XREF_PREV((_iter_b)))

#define ccxll_comp_geq_next(_iter_a, _iter_b)  (XREF_NEXT((_iter_a)) >=       \
                                                XREF_NEXT((_iter_b)))



/* ccxll iterators */


#define ccxll_iter_copy(_iter_dst, _iter_src)                                 \
                                                                              \
VOID_EXPR_                                                                    \
(                                                                             \
    *(_iter_dst) = *(_iter_src)                                               \
)


#define ccxll_iter_head(_iter)                                                \
                                                                              \
VOID_EXPR_                                                                    \
(                                                                             \
    (_iter)->prev.XOR = NULL,                                                 \
    (_iter)->curr.XOR = &((_iter)->cont->head.XOR),                           \
    (_iter)->next.XOR = &((_iter)->cont->head.stnl->XOR)                      \
)


#define ccxll_iter_tail(_iter)                                                \
                                                                              \
VOID_EXPR_                                                                    \
(                                                                             \
    (_iter)->next.XOR = NULL,                                                 \
    (_iter)->curr.XOR = &((_iter)->cont->tail.XOR),                           \
    (_iter)->prev.XOR = &((_iter)->cont->tail.stnl->XOR)                      \
)

```

```
549
550 #define ccxll_iter_begin(_iter)                                          \
551 (                                                                        \
552     (_iter)->prev.XOR =      &((_iter)->cont->head.XOR),                 \
553     (_iter)->curr.XOR =      &((_iter)->cont->head.stnl->XOR),           \
554     (_iter)->next.XOR = XOR2(&((_iter)->cont->head.XOR),                 \
555                              (_iter)->cont->head.stnl->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.stnl->XOR),           \
563     (_iter)->prev.XOR = XOR2(&((_iter)->cont->tail.XOR),                 \
564                              (_iter)->cont->tail.stnl->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

```
#ifndef OPENGC3_LIST_CCDLL_H
#define OPENGC3_LIST_CCDLL_H

#include "base.h"
#include "../base/pool.h"
#include "../base/misc.h"
#include "../base/snym.h"

#include <math.h>
#include <stddef.h>
#include <stdint.h>
#include <string.h>


/* ccdll create */


#define ccdll(elem_t)                                                       \
                                                                            \
        ccdll_extd(elem_t, 1, NORMAL)

#define ccdll_pckd(elem_t)                                                  \
                                                                            \
        ccdll_extd(elem_t, 1, PACKED)

#define ccdll_extd(elem_t, _n_iter, _ALIGN_)                                \
                                                                            \
        typedef ccdll_struct_extd(elem_t, _n_iter, _ALIGN_) *CCDLL;  CCDLL

#define ccdll_type(elem_t)                                                  \
                                                                            \
        typedef ccdll_struct_extd(elem_t, 1, NORMAL) *


#define ccdll_struct(elem_t)                                                \
                                                                            \
        ccdll_struct_extd(elem_t, 1, NORMAL)

#define ccdll_struct_pckd(elem_t)                                           \
                                                                            \
        ccdll_struct_extd(elem_t, 1, PACKED)

#define ccdll_struct_extd(elem_t, _n_iter, _ALIGN_)                         \
                                                                            \
    struct CCDLL_CONT                                                       \
    {                                                                       \
        int size,  last,  vcnt;                 /* size and node record */  \
        int start, ratio, thrsh;                /* block increment info */  \
```

```
                                                                        \
        struct CCDLL_NODE                                               \
        {   struct CCDLL_NODE *lnk[2];                                  \
            elem_t val;                             /* val with prv and nxt */  \
        }   *avsp, *pnode, head, tail, swap;        /* available space list */  \
                                                                        \
        struct CCDLL_BLCK                                               \
        {   struct CCDLL_BLCK *bprv, *bnxt;         /* points to prev block */  \
            int ncnt;                               /* the item of the node */  \
            PRAGMA_##_ALIGN_##_BGN                  /* packed pragma starts */  \
            struct CCDLL_NODE nodes[];              /* node structure array */  \
            PRAGMA_##_ALIGN_##_END                  /* the pragma ends here */  \
        }   *pool, *pblock;                         /* points to 1-st block */  \
                                                                        \
        struct CCDLL_ITER                                               \
        {   struct CCDLL_PTRS                                           \
            {   struct CCDLL_NODE *node;                                \
            }   curr;                               /* points to curr. node */  \
            struct CCDLL_CONT *cont;                /* points to ccdll body */  \
        }   (*itarr)[_n_iter], *_iter, **_it;                           \
                                                                        \
        struct CCDLL_CONT **_co;                    /* internal use _it _co */  \
                                                                        \
        int _it_base, _it_limit;                                        \
        int _co_base, _co_limit;                                        \
    }


/* ccdll initialize */


#define ccdll_init(_ccdll)                                              \
                                                                        \
        ccdll_init_extd(_ccdll,     16,      2,  65536)

#define ccdll_init_extd(_ccdll, _start, _ratio, _thrsh)                 \
                                                                        \
        cc_ll_init_extd(_ccdll, _start, _ratio, _thrsh, ccdll)


#define _ccdll_init(_ccdll_dst, _ccdll_src, _alloc)                     \
                                                                        \
        _ccdll_init_extd(_ccdll_dst, -1,     -1,     -1, _alloc)

#define _ccdll_init_extd(_ccdll, _start, _ratio, _thrsh, _alloc)        \
                                                                        \
        _cc_ll_init_extd(_ccdll, _start, _ratio, _thrsh, _alloc, ccdll)


```

```
#define _ccdll_init_core(_ccdll)                                        \
                                                                        \
        _cc_ll_init_core(_ccdll, ccdll)


#define _ccdll_init_seed(_ccdll)                                        \
                                                                        \
VOID_EXPR_                                                              \
(                                                                       \
    (_ccdll)->size = 0,                                                 \
    (_ccdll)->last = (_ccdll)->vcnt = 0,                                \
                                                                        \
    (_ccdll)->head.PRV = NULL,                                          \
    (_ccdll)->head.NXT = &((_ccdll)->tail),                             \
    (_ccdll)->tail.PRV = &((_ccdll)->head),                             \
    (_ccdll)->tail.NXT = NULL                                           \
)


#define _ccdll_init_info(_ccdll, _start, _ratio, _thrsh)                \
                                                                        \
        _cc_ll_init_info(_ccdll, _start, _ratio, _thrsh)


#define ccdll_iter_init(_iter, _ccdll)                                  \
                                                                        \
VOID_EXPR_                                                              \
(                                                                       \
    (_iter)->curr.node = NULL,                                          \
    (_iter)->cont = (_ccdll)                                            \
)


#define _ccdll_iter_init(_iter, _ccdll, _alloc)                         \
                                                                        \
        _cc_ll_iter_init(_iter, _ccdll, _alloc, ccdll)



/* ccdll destroy */


#define ccdll_free(_ccdll)  cc_ll_free(_ccdll)



/* ccdll access */


#define ccdll_front(_ccdll)  ((_ccdll)->head.NXT->val)
```

```
149
150 #define ccdll_back(_ccdll)   ((_ccdll)->tail.PRV->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.PRV == &((_ccdll)->head))
161
162
163
164 /* ccdll modifiers */
165
166
167 #define ccdll_push_front(_ccdll, _val) _ccdll_push(_ccdll, _val, head, NXT, PRV)
168
169 #define ccdll_push_back(_ccdll, _val)  _ccdll_push(_ccdll, _val, tail, PRV, 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, PRV)
182
183 #define ccdll_push_back_alloc(_ccdll)  _ccdll_push_alloc(_ccdll, tail, PRV, 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->PRV->NXT = (_ccdll)->pnode;                              \
195     (_ccdll)->pnode->NXT->PRV = (_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)) ||                             \
```

```
                    ccdll_iter_at_tail((_iter_i))))  break;                    \
                                                                                \
    _ccdll_move((_iter_p), (_iter_i));                                          \
                                                                                \
    (_iter_i)->cont->size--;                                                    \
    (_iter_i)->cont = (_iter_p)->cont;                                          \
    (_iter_i)->cont->size++;                                                    \
)

#define _ccdll_move(_iter_p, _iter_i)                                           \
                                                                                \
STATEMENT_                                                                      \
(                                                                               \
    (_iter_i)->curr.node->PRV->NXT = (_iter_i)->curr.node->NXT;                 \
    (_iter_i)->curr.node->NXT->PRV = (_iter_i)->curr.node->PRV;                 \
                                                                                \
    (_iter_i)->curr.node->PRV      = (_iter_p)->curr.node->PRV;                 \
    (_iter_i)->curr.node->NXT      = (_iter_p)->curr.node;                      \
                                                                                \
    (_iter_p)->curr.node->PRV      = (_iter_i)->curr.node;                      \
    (_iter_i)->curr.node->PRV->NXT = (_iter_i)->curr.node;                      \
)


#define ccdll_move_begin(_iter_a, _iter_b)                                      \
                                                                                \
STATEMENT_                                                                      \
(                                                                               \
    ccdll_iter_begin((_iter_a));                                                \
    ccdll_iter_begin((_iter_b));                                                \
                                                                                \
    ccdll_move((_iter_a), (_iter_b));                                           \
)


#define ccdll_move_range(_iter_p, _iter_l, _iter_r)                             \
                                                                                \
        ccdll_move_range_extd(_iter_p, _iter_l, _iter_r,    -1)

#define ccdll_move_range_extd(_iter_p, _iter_l, _iter_r, _dist)  /* [l, r) */   \
                                                                                \
STATEMENT_                                                                      \
(                                                                               \
    if (_unlikely(ccdll_iter_at_head ((_iter_p)) ||                             \
                  ccdll_iter_at_head ((_iter_l))))  break;                      \
                                                                                \
    if (_unlikely((_iter_l)->curr.node == (_iter_r)->curr.node))  break;        \
                                                                                \
    if (_unlikely((_iter_l)->cont != (_iter_r)->cont))  break;                  \
                                                                                \
```

```
    if (_unlikely((_iter_p)->cont != (_iter_l)->cont))                          \
    {                                                                           \
        int _dist_m = (_dist);                                                  \
                                                                                \
        if (_dist_m <  0)                                                       \
            ccdll_iter_distance((_iter_l), (_iter_r), &_dist_m);                \
                                                                                \
        if (_dist_m <= 0)  break;                                               \
                                                                                \
        (_iter_p)->cont->size += _dist_m;                                       \
        (_iter_l)->cont->size -= _dist_m;                                       \
        (_iter_l)->cont = (_iter_p)->cont;                                      \
    }                                                                           \
                                                                                \
    void *_pbup = (_iter_r)->curr.node;                                         \
                                                                                \
    (_iter_r)->curr.node = (_iter_r)->curr.node->PRV;                           \
                                                                                \
    (_iter_l)->curr.node->PRV->NXT = (_iter_r)->curr.node->NXT;                 \
    (_iter_r)->curr.node->NXT->PRV = (_iter_l)->curr.node->PRV;                 \
                                                                                \
    (_iter_l)->curr.node->PRV      = (_iter_p)->curr.node->PRV;                 \
    (_iter_p)->curr.node->PRV->NXT = (_iter_l)->curr.node;                      \
                                                                                \
    (_iter_r)->curr.node->NXT      = (_iter_p)->curr.node;                      \
    (_iter_p)->curr.node->PRV      = (_iter_r)->curr.node;                      \
                                                                                \
    (_iter_r)->curr.node = _pbup;                                               \
)


#define  ccdll_merge(_ccdll_d, _ccdll_s)                                        \
                                                                                \
         ccdll_merge_extd(_ccdll_d, _ccdll_s, DLEQ)

#define  ccdll_merge_extd(_ccdll_d, _ccdll_s, _leq)                             \
                                                                                \
         cc_ll_merge_extd(_ccdll_d, _ccdll_s, _leq, ccdll, )

#define _ccdll_merge_extd(_iter_l, _iter_m, _iter_r, _leq)                      \
                                                                                \
        _cc_ll_merge_extd(_iter_l, _iter_m, _iter_r, _leq, ccdll, )


#define  ccdll_merge_range(_iter_l, _iter_m, _iter_r)                           \
                                                                                \
         ccdll_merge_range_extd(_iter_l, _iter_m, _iter_r, DLEQ)

#define  ccdll_merge_range_extd(_iter_l, _iter_m, _iter_r, _leq)                \
                                                                                \
```

```
          cc_ll_merge_range_extd(_iter_l, _iter_m, _iter_r, _leq, ccdll, )

#define _ccdll_merge_range_extd(_iter_l, _iter_m, _iter_r, _iter_x, _leq)      \
                                                                              \
        _cc_ll_merge_range_extd(_iter_l, _iter_m, _iter_r, _iter_x, _leq, ccdll)


#define  ccdll_sort(_ccdll)                                                   \
                                                                              \
          ccdll_sort_extd(_ccdll, DLEQ)

#define  ccdll_sort_extd(_ccdll, _leq)                                        \
                                                                              \
          cc_ll_sort_extd(_ccdll, _leq, ccdll, )

#define _ccdll_sort_extd(_ccdll, _carry, _pbuck, _iter_a, _iter_b, _leq)       \
                                                                              \
        _cc_ll_sort_extd(_ccdll, _carry, _pbuck, _iter_a, _iter_b, _leq, ccdll,)


/* ccdll comparator */

#define ccdll_comp_leq(_iter_a, _iter_b)       (DREF((_iter_a)) <=            \
                                                DREF((_iter_b)))

#define ccdll_comp_leq_prev(_iter_a, _iter_b)  (DREF_PREV((_iter_a)) <=       \
                                                DREF_PREV((_iter_b)))

#define ccdll_comp_leq_next(_iter_a, _iter_b)  (DREF_NEXT((_iter_a)) <=       \
                                                DREF_NEXT((_iter_b)))

#define ccdll_comp_geq(_iter_a, _iter_b)       (DREF((_iter_a)) >=            \
                                                DREF((_iter_b)))

#define ccdll_comp_geq_prev(_iter_a, _iter_b)  (DREF_PREV((_iter_a)) >=       \
                                                DREF_PREV((_iter_b)))

#define ccdll_comp_geq_next(_iter_a, _iter_b)  (DREF_NEXT((_iter_a)) >=       \
                                                DREF_NEXT((_iter_b)))


/* ccdll iterators */

#define ccdll_iter_copy(_iter_dst, _iter_src)                                 \
                                                                              \
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.PRV)                  \
483 )
484
485
486 #define ccdll_iter_at_head(_iter)   ( (_iter)->curr.node->PRV == 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->PRV ==        \
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 (                                                                   \
```

```
    ccdll_iter_at_tail((_iter)) ? (NULL) :                                  \
    ((_iter)->curr.node = (_iter)->curr.node->NXT)->NXT                     \
)


#define ccdll_iter_decr(_iter)                                              \
(                                                                           \
    ccdll_iter_at_head((_iter)) ? (NULL) :                                  \
    ((_iter)->curr.node = (_iter)->curr.node->PRV)->PRV                     \
)


#define ccdll_iter_advance(_iter, _diff)                                    \
                                                                            \
STATEMENT_                                                                  \
(                                                                           \
    int _len = (_diff);                                                     \
                                                                            \
         if (_len > 0)  while (ccdll_iter_incr((_iter)) && --_len);         \
    else if (_len < 0)  while (ccdll_iter_decr((_iter)) && ++_len);         \
)


#define ccdll_iter_distance(_iter_a, _iter_b, _pdist)                       \
                                                                            \
        cc_ll_iter_distance(_iter_a, _iter_b, _pdist, ccdll)



/* ccdll traversor */


#define CCDLL_INCR(_iter)  CC_LL_INCR(_iter, ccdll)

#ifndef CC_STRICT

#define CCDLL_INCR_AUTO(_pval, _ccdll)                                      \
                                                                            \
        CCDLL_INCR_EXTD(_pval, _ccdll, (void)0)

#define CCDLL_INCR_EXTD(_pval, _ccdll, ...)                                 \
                                                                            \
        CC_LL_INCR_EXTD(_pval, _ccdll, ccdll, __VA_ARGS__)

#endif // CC_STRICT


#define CCDLL_DECR(_iter)  CC_LL_DECR(_iter, ccdll)

#ifndef CC_STRICT
```

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

1.1.4 list/ccsll.h

```
#ifndef OPENGC3_LIST_CCSLL_H
#define OPENGC3_LIST_CCSLL_H

#include "base.h"
#include "../base/pool.h"
#include "../base/misc.h"
#include "../base/snym.h"

#include <math.h>
#include <stddef.h>
#include <stdint.h>
#include <string.h>


/* ccsll create */


#define ccsll(elem_t)                                                       \
                                                                            \
        ccsll_extd(elem_t, 1, NORMAL)

#define ccsll_pckd(elem_t)                                                  \
                                                                            \
        ccsll_extd(elem_t, 1, PACKED)

#define ccsll_extd(elem_t, _n_iter, _ALIGN_)                                \
                                                                            \
        typedef ccsll_struct_extd(elem_t, _n_iter, _ALIGN_) *CCSLL;  CCSLL

#define ccsll_type(elem_t)                                                  \
                                                                            \
        typedef ccsll_struct_extd(elem_t, 1, NORMAL) *


#define ccsll_struct(elem_t)                                                \
                                                                            \
        ccsll_struct_extd(elem_t, 1, NORMAL)

#define ccsll_struct_pckd(elem_t)                                           \
                                                                            \
        ccsll_struct_extd(elem_t, 1, PACKED)

#define ccsll_struct_extd(elem_t, _n_iter, _ALIGN_)                         \
                                                                            \
    struct CCSLL_CONT                                                       \
    {                                                                       \
        int size,  last,  vcnt;                 /* size and node record */  \
        int start, ratio, thrsh;                /* block increment info */  \
```

```
                                                                        \
        struct CCSLL_NODE                                               \
        {   struct CCSLL_NODE *lnk[1];                                  \
            elem_t val;                             /* val with a next link */  \
        }   *avsp, *pnode, head, tail, swap;        /* available space list */  \
                                                                        \
        struct CCSLL_BLCK                                               \
        {   struct CCSLL_BLCK *bprv, *bnxt;         /* points to prev block */  \
            int ncnt;                               /* the item of the node */  \
            PRAGMA_##_ALIGN_##_BGN                  /* packed pragma starts */  \
            struct CCSLL_NODE nodes[];              /* node structure array */  \
            PRAGMA_##_ALIGN_##_END                  /* the pragma ends here */  \
        }   *pool, *pblock;                         /* points to 1-st block */  \
                                                                        \
        struct CCSLL_ITER                                               \
        {   struct CCSLL_PTRS                                           \
            {   struct CCSLL_NODE *node;                                \
            }   curr;                               /* points to curr  node */  \
            struct CCSLL_CONT *cont;                /* points to ccsll body */  \
        }   (*itarr)[_n_iter], *_iter, **_it;                           \
                                                                        \
        struct CCSLL_CONT **_co;                    /* internal use _it _co */  \
                                                                        \
        int _it_base, _it_limit;                                        \
        int _co_base, _co_limit;                                        \
    }


/* ccsll initialize */


#define ccsll_init(_ccsll)                                              \
                                                                        \
        ccsll_init_extd(_ccsll,     16,      2,  65536)

#define ccsll_init_extd(_ccsll, _start, _ratio, _thrsh)                 \
                                                                        \
        cc_ll_init_extd(_ccsll, _start, _ratio, _thrsh, ccsll)


#define _ccsll_init(_ccsll_dst, _ccsll_src, _alloc)                     \
                                                                        \
        _ccsll_init_extd(_ccsll_dst, -1,     -1,     -1, _alloc)

#define _ccsll_init_extd(_ccsll, _start, _ratio, _thrsh, _alloc)        \
                                                                        \
        _cc_ll_init_extd(_ccsll, _start, _ratio, _thrsh, _alloc, ccsll)


```

```
#define _ccsll_init_core(_ccsll)                                                \
                                                                                \
        _cc_ll_init_core(_ccsll, ccsll)


#define _ccsll_init_seed(_ccsll)                                                \
                                                                                \
VOID_EXPR_                                                                      \
(                                                                               \
    (_ccsll)->size = 0,                                                         \
    (_ccsll)->last = (_ccsll)->vcnt = 0,                                        \
                                                                                \
    (_ccsll)->head.NXT = &((_ccsll)->tail),                                     \
    (_ccsll)->tail.NXT = NULL                                                   \
)


#define _ccsll_init_info(_ccsll, _start, _ratio, _thrsh)                        \
                                                                                \
        _cc_ll_init_info(_ccsll, _start, _ratio, _thrsh)


#define ccsll_iter_init(_iter, _ccsll)                                          \
                                                                                \
VOID_EXPR_                                                                      \
(                                                                               \
    (_iter)->curr.node = NULL,                                                  \
    (_iter)->cont = (_ccsll)                                                    \
)


#define _ccsll_iter_init(_iter, _ccsll, _alloc)                                 \
                                                                                \
        _cc_ll_iter_init(_iter, _ccsll, _alloc, ccsll)



/* ccsll destroy */


#define ccsll_free(_ccsll)  cc_ll_free(_ccsll)



/* ccsll access */


#define ccsll_front(_ccsll)  ((_ccsll)->head.NXT->val)


```

```
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 ccsll_insert(_iter, _val)                                       \
201                                                                         \
202 STATEMENT_                                                              \
203 (                                                                       \
204     if (ccsll_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 ccsll_erase(_iter)                                              \
215                                                                         \
216 STATEMENT_                                                              \
217 (                                                                       \
218     if (ccsll_iter_at_head((_iter)) || ccsll_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 ccsll_swap(_ccsll_a, _ccsll_b)                                  \
229                                                                         \
230 STATEMENT_                                                              \
231 (                                                                       \
232     XOR2_SWAP((_ccsll_a), (_ccsll_b));                                  \
233 )
234
235
236 #define ccsll_resize(_ccsll, _items, _val)                              \
237                                                                         \
238 STATEMENT_                                                              \
239 (                                                                       \
240     int _size = ccsll_size((_ccsll)) - (_items);                        \
241                                                                         \
242          if (_size > 0)  while(_size--)  ccsll_pop_front ((_ccsll));    \
243     else if (_size < 0)  while(_size++)  ccsll_push_front((_ccsll), (_val));    \
244 )
245
246
247 #define ccsll_clear(_ccsll)                                             \
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)                         \
```

```
                                                                                \
        ccsll_move_range_extd(_iter_p, _iter_l, _iter_r,    -1)

#define ccsll_move_range_extd(_iter_p, _iter_l, _iter_r, _dist)  /* (l, r] */  \
                                                                                \
STATEMENT_                                                                      \
(                                                                               \
    if (_unlikely(ccsll_iter_at_tail((_iter_p)) ||                              \
                  ccsll_iter_at_tail((_iter_r))))  break;                       \
                                                                                \
    if (_unlikely((_iter_l)->curr.node == (_iter_r)->curr.node))  break;        \
                                                                                \
    if (_unlikely((_iter_l)->cont != (_iter_r)->cont))  break;                  \
                                                                                \
    if (_unlikely((_iter_p)->cont != (_iter_r)->cont))                          \
    {                                                                           \
        int _dist_m = (_dist);                                                  \
                                                                                \
        if (_dist_m <  0)                                                       \
            ccsll_iter_distance((_iter_l), (_iter_r), &_dist_m);                \
                                                                                \
        if (_dist_m <= 0)  break;                                               \
                                                                                \
        (_iter_p)->cont->size += _dist_m;                                       \
        (_iter_r)->cont->size -= _dist_m;                                       \
        (_iter_r)->cont = (_iter_p)->cont;                                      \
    }                                                                           \
                                                                                \
    void *_pbup = (_iter_p)->curr.node->NXT;                                    \
                                                                                \
    (_iter_p)->curr.node->NXT = (_iter_l)->curr.node->NXT;                      \
    (_iter_l)->curr.node->NXT = (_iter_r)->curr.node->NXT;                      \
    (_iter_r)->curr.node->NXT =  _pbup;                                         \
)


#define  ccsll_merge(_ccsll_d, _ccsll_s)                                        \
                                                                                \
         ccsll_merge_extd(_ccsll_d, _ccsll_s, SLEQ_NEXT)

#define  ccsll_merge_extd(_ccsll_d, _ccsll_s, _leq)                             \
                                                                                \
         cc_ll_merge_extd(_ccsll_d, _ccsll_s, _leq, ccsll, )

#define _ccsll_merge_extd(_iter_l, _iter_m, _iter_r, _leq)                      \
                                                                                \
STATEMENT_                                                                      \
(                                                                               \
    if (_unlikely((_iter_l)->cont == (_iter_m)->cont ||                         \
                  (_iter_l)->cont == (_iter_r)->cont ||                         \
```

```
                    (_iter_m)->cont != (_iter_r)->cont))  break;                \
                                                                                \
    ccsll_iter_head((_iter_l));                                                 \
    ccsll_iter_head((_iter_m));                                                 \
    ccsll_iter_head((_iter_r));                                                 \
                                                                                \
    for (register int _len = 0; ; _len = 0)                                     \
    {                                                                           \
        while (!(ccsll_iter_at_end((_iter_l))) && _leq((_iter_l), (_iter_m)))  \
            ((void)ccsll_iter_incr((_iter_l)));                                 \
                                                                                \
        while (!(ccsll_iter_at_end((_iter_r))) &&                               \
                (ccsll_iter_at_end((_iter_l)) || !_leq((_iter_l), (_iter_r)))) \
            ((void)ccsll_iter_incr((_iter_r)), ++_len);                         \
                                                                                \
        ccsll_move_range_extd((_iter_l), (_iter_m), (_iter_r), _len);           \
                                                                                \
        if (ccsll_iter_at_end((_iter_m)))  break;                               \
                                                                                \
        ccsll_iter_copy((_iter_l), (_iter_r));                                  \
        ccsll_iter_init((_iter_r), (_iter_m)->cont) ;                           \
        ccsll_iter_head((_iter_r));                                             \
    }                                                                           \
)


#define  ccsll_sort(_ccsll)                                                     \
                                                                                \
         ccsll_sort_extd(_ccsll, SLEQ_NEXT)

#define  ccsll_sort_extd(_ccsll, _leq)                                          \
                                                                                \
         cc_ll_sort_extd(_ccsll, _leq, ccsll, )

#define _ccsll_sort_extd(_ccsll, _carry, _pbuck, _iter_a, _iter_b, _leq)       \
                                                                                \
        _cc_ll_sort_extd(_ccsll, _carry, _pbuck, _iter_a, _iter_b, _leq, ccsll,)



/* ccsll comparator */


#define ccsll_comp_leq(_iter_a, _iter_b)       (SREF((_iter_a)) <=             \
                                                SREF((_iter_b)))

#define ccsll_comp_leq_next(_iter_a, _iter_b)  (SREF_NEXT((_iter_a)) <=        \
                                                SREF_NEXT((_iter_b)))

#define ccsll_comp_geq(_iter_a, _iter_b)       (SREF((_iter_a)) >=             \
```

```
                                                SREF((_iter_b)))

#define ccsll_comp_geq_next(_iter_a, _iter_b)  (SREF_NEXT((_iter_a)) >=        \
                                                SREF_NEXT((_iter_b)))



/* ccsll iterators */


#define ccsll_iter_copy(_iter_dst, _iter_src)                                  \
                                                                               \
VOID_EXPR_                                                                     \
(                                                                              \
    *(_iter_dst) = *(_iter_src)                                                \
)


#define ccsll_iter_head(_iter)                                                 \
                                                                               \
VOID_EXPR_                                                                     \
(                                                                              \
    (_iter)->curr.node = &((_iter)->cont->head)                                \
)


#define ccsll_iter_tail(_iter)                                                 \
                                                                               \
VOID_EXPR_                                                                     \
(                                                                              \
    (_iter)->curr.node = &((_iter)->cont->tail)                                \
)


#define ccsll_iter_begin(_iter)                                                \
                                                                               \
VOID_EXPR_                                                                     \
(                                                                              \
    (_iter)->curr.node =  ((_iter)->cont->head.NXT)                            \
)


#define ccsll_iter_at_head(_iter)   ( (_iter)->curr.node ==                    \
                                    &((_iter)->cont->head) )

#define ccsll_iter_at_tail(_iter)   ( (_iter)->curr.node ==                    \
                                    &((_iter)->cont->tail) )

#define ccsll_iter_at_begin(_iter)  ( (_iter)->curr.node ==                    \
                                      (_iter)->cont->head.NXT )
```

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

1.1.5 list/extd-base.h

```
#ifndef OPENGC3_LIST_EXTD_BASE_H
#define OPENGC3_LIST_EXTD_BASE_H


/* cc_ll integrity */


#define cc_ll_is_sorted_extd(_cc_ll, _leq, _ptrue, _ll_)                        \
                                                                                \
STATEMENT_                                                                      \
(                                                                               \
    (*(_ptrue)) = 1;                                                            \
                                                                                \
    if (_ll_##_size((_cc_ll)) <= 1)  break;                                     \
                                                                                \
    _it_init((_cc_ll), 2, _base_s3, _ll_);                                      \
                                                                                \
    _ll_##_iter_head (_it_((_cc_ll), _base_s3, 0));                             \
    _ll_##_iter_begin(_it_((_cc_ll), _base_s3, 1));                             \
                                                                                \
    while (1)                                                                   \
    {                                                                           \
        (void)_ll_##_iter_incr(_it_((_cc_ll), _base_s3, 0));                    \
        (void)_ll_##_iter_incr(_it_((_cc_ll), _base_s3, 1));                    \
                                                                                \
        if (!(_leq(_it_((_cc_ll), _base_s3, 0), _it_((_cc_ll), _base_s3, 1)))) \
        {                                                                       \
            (*(_ptrue)) = 0;  break;                                            \
        }                                                                       \
                                                                                \
        if (_ll_##_iter_at_end(_it_((_cc_ll), _base_s3, 1)))  break;            \
    }                                                                           \
                                                                                \
    _it_clear((_cc_ll), 2);                                                     \
)


#define cc_ll_is_robust(_cc_ll, _ptrue, _ll_, _LL_)          /* INCOMPLETE */  \
                                                                                \
STATEMENT_                                                                      \
(                                                                               \
    int _size = 0;                                                              \
                                                                                \
    _LL_##_INCR((_cc_ll)->_iter)  _size++;                                      \
                                                                                \
    *(_ptrue) = !!(_ll_##_size((_cc_ll)) == _size);                             \
)

```

```
49
50
51  #endif
```

1.1.6 list/extd-ccxll.h

```
#ifndef OPENGC3_LIST_EXTD_CCXLL_H
#define OPENGC3_LIST_EXTD_CCXLL_H

#include "ccxll.h"
#include "extd-base.h"
#include "../vect/array.h"


/* ccxll operations extended */


#define  ccxll_sort_destruct(_ccxll) /* SIZE OF MEMORY POOL BLOCKS MATTERS */  \
                                                                               \
         ccxll_sort_destruct_extd(_ccxll, XLEQ, ACMP)

#define  ccxll_sort_destruct_extd(_ccxll, _leq, _cmp)                          \
                                                                               \
STATEMENT_                                                                     \
(                                                                              \
    int _sbup = ccxll_size((_ccxll));                                          \
                                                                               \
    (_ccxll)->pblock = (_ccxll)->pool;                                         \
                                                                               \
    while ((_ccxll)->pblock != NULL)                                           \
    {                                                                          \
        int _lo = ((_ccxll)->pblock->bnxt == NULL) ? (_ccxll)->vcnt : 0;       \
        int _hi = ((_ccxll)->pblock->ncnt - 1);                                \
                                                                               \
        array_sort((_ccxll)->pblock->nodes, &(_ccxll)->swap, _lo, _hi, _cmp);  \
                                                                               \
        (_ccxll)->pool   = (_ccxll)->pblock;                                   \
        (_ccxll)->pblock = (_ccxll)->pblock->bprv;                             \
    }                                                                          \
                                                                               \
    _ccxll_init_seed((_ccxll));                                                \
                                                                               \
    while (_sbup-- > 0)                                                        \
        ccxll_push_front_alloc((_ccxll));                                      \
                                                                               \
    ccxll_block_merge_extd((_ccxll), _leq);                                    \
)


#define  ccxll_block_merge_extd(_ccxll, _leq)                                  \
                                                                               \
STATEMENT_                                                                     \
(                                                                              \
    _it_init((_ccxll), 3, _base_s1, ccxll);                                    \
```

```
                                                                                \
    _ccxll_block_merge_extd((_ccxll), _it_((_ccxll), _base_s1, 0),              \
                                      _it_((_ccxll), _base_s1, 1),              \
                                      _it_((_ccxll), _base_s1, 2), _leq);       \
                                                                                \
    _it_clear((_ccxll), 3);                                                     \
)


#define _ccxll_block_merge_extd(_ccxll, _iter_l, _iter_m, _iter_r, _leq)        \
                                                                                \
STATEMENT_                                                                      \
(                                                                               \
    if ((_ccxll)->pool == NULL)  break;                                         \
                                                                                \
    (_ccxll)->pblock = (_ccxll)->pool;                                          \
                                                                                \
    ccxll_iter_begin((_iter_r));                                                \
                                                                                \
    while ((_ccxll)->pblock != NULL)                                            \
    {                                                                           \
        ccxll_iter_begin((_iter_l));                                            \
        ccxll_iter_copy ((_iter_m), (_iter_r));                                 \
                                                                                \
       _ccxll_iter_block((_ccxll)->pblock->bprv, (_iter_r));                    \
        ccxll_merge_range_extd((_iter_l), (_iter_m), (_iter_r), _leq);          \
                                                                                \
        (_ccxll)->pblock = (_ccxll)->pblock->bprv;                              \
    }                                                                           \
)


#define _ccxll_iter_block(_pblock, _iter)                                       \
                                                                                \
STATEMENT_                                                                      \
(                                                                               \
    if ((_pblock) == NULL)                                                      \
        ccxll_iter_tail((_iter));                                               \
    else if ((_pblock)->bnxt == NULL)                                           \
        ccxll_iter_begin((_iter));                                              \
    else                                                                        \
    {                                                                           \
        (_iter)->curr.XOR = &(_pblock)->nodes[0];                               \
        (_iter)->prev.XOR = &(_pblock)->bnxt->nodes[(_pblock)->bnxt->ncnt - 1];\
        (_iter)->next.XOR = XOR2((_iter)->curr.node->XOR, (_iter)->prev.XOR);  \
    }                                                                           \
)
```

```
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-ccdll.h

```
#ifndef OPENGC3_LIST_EXTD_CCDLL_H
#define OPENGC3_LIST_EXTD_CCDLL_H

#include "ccdll.h"
#include "extd-base.h"
#include "extd-ccsll.h"


/* ccdll operations extended */


#define  ccdll_sort_prefetch(_ccdll)                                          \
                                                                              \
         ccdll_sort_prefetch_extd(_ccdll, SLEQ_NEXT)

#define  ccdll_sort_prefetch_extd(_ccdll, _leq)                               \
                                                                              \
         cc_ll_sort_extd         (_ccdll, _leq, ccdll, _prefetch)

#define _ccdll_sort_prefetch_extd(_ccdll,  _carry,  _pbuck,                   \
                                  _iter_a, _iter_b, _leq)                     \
        _cc_ll_sort_extd         (_ccdll,  _carry,  _pbuck,                   \
                                  _iter_a, _iter_b, _leq,  ccsll, _prefetch)


/* ccdll iterators extended */


#define ccdll_iter_incr_prefetch(_iter, _parr, _pofs)                         \
(                                                                             \
    *((_parr)[*(_pofs)]) =  ((_iter)->curr.node),                             \
      (_parr)[*(_pofs)]  = &((_iter)->curr.node->PRV),                        \
     *(_pofs) = (*(_pofs) + 1) % 64,                                          \
    __builtin_prefetch((_iter)->curr.node->PRV),                              \
    ccsll_iter_at_tail((_iter)) ? (NULL) :                                    \
    ((_iter)->curr.node = (_iter)->curr.node->NXT)->NXT                       \
)


/* ccdll integrity */


#define ccdll_is_sorted(_ccdll, _ptrue)                                       \
                                                                              \
         ccdll_is_sorted_extd(_ccdll, DLEQ, _ptrue)

```

```
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

```
#ifndef OPENGC3_LIST_EXTD_CCSLL_H
#define OPENGC3_LIST_EXTD_CCSLL_H

#include "ccsll.h"
#include "extd-base.h"
#include "extd-ccdll.h"


/* ccsll operations extended */


#define  ccsll_merge_prefetch(_ccsll_d, _ccsll_s)                              \
                                                                               \
         ccsll_merge_prefetch_extd(_ccsll_d, _ccsll_s, SLEQ_NEXT)

#define  ccsll_merge_prefetch_extd(_ccsll_d, _ccsll_s, _leq)                   \
                                                                               \
         cc_ll_merge_extd         (_ccsll_d, _ccsll_s, _leq, ccsll, _prefetch)

#define _ccsll_merge_prefetch_extd(_iter_l, _iter_m, _iter_r, _leq)            \
                                                                               \
STATEMENT_                                                                     \
(                                                                              \
    if (_unlikely((_iter_l)->cont == (_iter_m)->cont ||                        \
                  (_iter_l)->cont == (_iter_r)->cont ||                        \
                  (_iter_m)->cont != (_iter_r)->cont))  break;                 \
                                                                               \
    int _ofs = 0;                                                              \
    __typeof__((_iter_l)->cont->pnode) _write, *_queue[64];                    \
                                                                               \
    for (int _idx = 0; _idx < 64; _idx++)                                      \
        (_queue)[_idx] = &_write;                                              \
                                                                               \
    ccsll_iter_head((_iter_l));                                                \
    ccsll_iter_head((_iter_m));                                                \
    ccsll_iter_head((_iter_r));                                                \
                                                                               \
    for (register int _len = 0; ; _len = 0)                                    \
    {                                                                          \
        while (!(ccsll_iter_at_end((_iter_l))) && _leq((_iter_l), (_iter_m)))  \
            ((void)ccdll_iter_incr_prefetch((_iter_l), _queue, &_ofs));        \
                                                                               \
        while (!(ccsll_iter_at_end((_iter_r))) &&                              \
                (ccsll_iter_at_end((_iter_l)) || !_leq((_iter_l), (_iter_r)))) \
            ((void)ccdll_iter_incr_prefetch((_iter_r), _queue, &_ofs), ++_len);\
                                                                               \
        ccsll_move_range_extd((_iter_l), (_iter_m), (_iter_r), _len);          \
                                                                               \
```

```
        if (ccsll_iter_at_end((_iter_m)))  break;                              \
                                                                               \
        ccsll_iter_copy((_iter_l), (_iter_r));                                 \
        ccsll_iter_init((_iter_r), (_iter_m)->cont) ;                          \
        ccsll_iter_head((_iter_r));                                            \
    }                                                                          \
)



/* ccsll integrity */


#define ccsll_is_sorted(_ccsll, _ptrue)                                        \
                                                                               \
        ccsll_is_sorted_extd(_ccsll, SLEQ, _ptrue)

#define ccsll_is_sorted_extd(_ccsll, _leq, _ptrue)                             \
                                                                               \
        cc_ll_is_sorted_extd(_ccsll, _leq, _ptrue, ccsll)


#define ccsll_is_robust(_ccsll, _ptrue)                                        \
                                                                               \
        cc_ll_is_robust(_ccsll, _ptrue, ccsll, CCSLL)



#endif
```

1.2 base

1.2.1 base/mesg.h

```
#ifndef OPENGC3_BASE_MESG_H
#define OPENGC3_BASE_MESG_H

#include <stdio.h>
#include <stdlib.h>

#include "misc.h"


/* error and fatal messages */


static const char CC_ERROR_MSG_MEMORY_LEAK[] = "Potential Memory Leak.";
static const char CC_ERROR_MSG_DOUBLE_FREE[] = "Potential Double Free.";

#define CC_ERROR(CC_ERROR_MSG)                                                  \
                                                                                \
STATEMENT_                                                                      \
(                                                                               \
    fprintf(stderr, "OpenGC^3::" "ERROR: %s\n", CC_ERROR_MSG);                  \
)


static const char CC_FATAL_MSG_MALLOC_FAIL[] = "Memory Allocation Failure.";
static const int  CC_FATAL_MSG_MALLOC_FAIL_EXITCODE = -1;

#define CC_FATAL(CC_FATAL_MSG)                                                  \
                                                                                \
STATEMENT_                                                                      \
(                                                                               \
    fprintf(stderr, "OpenGC^3::" "FATAL: %s\n", CC_FATAL_MSG);                  \
    exit(CC_FATAL_MSG##_EXITCODE);                                              \
)



#endif
```

1.2.2 base/misc.h

```
#ifndef OPENGC3_BASE_MISC_H
#define OPENGC3_BASE_MISC_H


/* syntax wrapper */

#define STATEMENT_(...)   do {__VA_ARGS__} while (0)
#define VOID_EXPR_(...)   ((__VA_ARGS__), ((void)0))


/* general macros */

#define MIN_2(A, B)  ((A) < (B) ? (A) : (B))
#define BITSOF(VAR)  (sizeof(VAR) * CHAR_BIT)
#define ELEMOF(ARR)  (sizeof(ARR) / sizeof(ARR[0]))
#define UMAXOF(VAR)  (~UINT64_C(0) >> (64 - BITSOF(VAR)))


/* compiler pragmas */

#define PRAGMA_NORMAL_BGN
#define PRAGMA_NORMAL_END

#define PRAGMA_PACKED_BGN  _Pragma("pack(push, 1)")
#define PRAGMA_PACKED_END  _Pragma("pack(pop)"    )


/* compiler extensions */

#ifndef CC_STRICT
    #define _unlikely(_expr)  (__builtin_expect(!!(_expr), 0))
    #define _prefetch(_addr)  (__builtin_prefetch((_addr)))
    #define _it_(_cont, _iter, _offset)  (&(_iter)[(_offset)])
    #define _co_(_cont, _base, _offset)  ((_cont)->_co[(_base) + (_offset)])
#else
    #define _unlikely(_expr)  (_expr)
    #define _prefetch(_addr)  (_addr)
    #define _it_(_cont, _base, _offset)  ((_cont)->_it[(_base) + (_offset)])
    #define _co_(_cont, _base, _offset)  ((_cont)->_co[(_base) + (_offset)])
#endif // CC_STRICT


/* pointer layout */

#define XOR  lnk[0]
#define NXT  lnk[0]
#define PRV  lnk[1]

```

```
#define PRN  lnk[0]
#define LFT  lnk[1]
#define RGH  lnk[2]


/* append line ID */

#define ADDID APPENDLINE
#define CONCATLINE(N, L)  N ## _ ## L
#define EXPANDLINE(N, L)  CONCATLINE(N, L)
#define APPENDLINE(NAME)  EXPANDLINE(NAME, __LINE__)

#define CCDLL       ADDID(CCDLL)
#define CCDLL_CONT  ADDID(CCDLL_CONT)
#define CCDLL_NODE  ADDID(CCDLL_NODE)
#define CCDLL_BLCK  ADDID(CCDLL_BLCK)
#define CCDLL_ITER  ADDID(CCDLL_ITER)
#define CCDLL_PTRS  ADDID(CCDLL_PTRS)

#define CCSLL       ADDID(CCSLL)
#define CCSLL_CONT  ADDID(CCSLL_CONT)
#define CCSLL_NODE  ADDID(CCSLL_NODE)
#define CCSLL_BLCK  ADDID(CCSLL_BLCK)
#define CCSLL_ITER  ADDID(CCSLL_ITER)
#define CCSLL_PTRS  ADDID(CCSLL_PTRS)

#define CCXLL       ADDID(CCXLL)
#define CCXLL_CONT  ADDID(CCXLL_CONT)
#define CCXLL_NODE  ADDID(CCXLL_NODE)
#define CCXLL_BLCK  ADDID(CCXLL_BLCK)
#define CCXLL_ITER  ADDID(CCXLL_ITER)
#define CCXLL_HDTL  ADDID(CCXLL_HDTL)
#define CCXLL_PTRS  ADDID(CCXLL_PTRS)

#define CCGBT       ADDID(CCGBT)
#define CCGBT_CONT  ADDID(CCGBT_CONT)
#define CCGBT_NODE  ADDID(CCGBT_NODE)
#define CCGBT_BLCK  ADDID(CCGBT_BLCK)
#define CCGBT_ITER  ADDID(CCGBT_ITER)
#define CCGBT_PTRS  ADDID(CCGBT_PTRS)

#define CCARR       ADDID(CCARR)
#define CCARR_CONT  ADDID(CCARR_CONT)



#endif
```

1.2.3 base/pool.h

```
#ifndef OPENGC3_BASE_POOL_H
#define OPENGC3_BASE_POOL_H

#include "mesg.h"
#include "misc.h"

#include <stdio.h>
#include <string.h>


/* safe allocation */


#define _safe_alloc(_void_ptr, _alloc_bytes)                                    \
                                                                                \
STATEMENT_                                                                      \
(                                                                               \
    if ((_void_ptr) != NULL)                                                    \
        CC_ERROR(CC_ERROR_MSG_MEMORY_LEAK);                                     \
                                                                                \
    if (((_void_ptr) = malloc((_alloc_bytes))) == NULL)                         \
        CC_FATAL(CC_FATAL_MSG_MALLOC_FAIL);                                     \
)


#define _safe_free(_void_ptr)                                                   \
                                                                                \
STATEMENT_                                                                      \
(                                                                               \
    if ((_void_ptr) == NULL)                                                    \
        CC_ERROR(CC_ERROR_MSG_DOUBLE_FREE);                                     \
                                                                                \
    free((_void_ptr));                                                          \
    (_void_ptr) = NULL;                                                         \
)



/* cont / iter */


#define _cont_alloc(_cont)  _safe_alloc((_cont), sizeof(*(_cont)))

#define _cont_free(_cont)   _safe_free ((_cont))

#define _iter_alloc(_iter)  _safe_alloc((_iter), sizeof(*(_iter)))

#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)(ELEMOF(*(_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++)                       \
```

```
        _iter_free((_cont)->_it[_idx_it]);                                  \
                                                                            \
    _auxr_free((_cont), _it);                                               \
)


#define _co_free(_cont)                                                     \
                                                                            \
STATEMENT_                                                                  \
(                                                                           \
    int _co_total = _itco_total((_cont), _co);                              \
                                                                            \
    for (int _idx_co = 0; _idx_co < _co_total; _idx_co++)                   \
    {                                                                       \
        _it_free   ((_cont)->_co[_idx_co]);                                 \
        _block_free((_cont)->_co[_idx_co]);                                 \
        _cont_free ((_cont)->_co[_idx_co]);                                 \
    }                                                                       \
                                                                            \
    _auxr_free((_cont), _co);                                               \
)



/* auxr management */


#define _auxr_alloc(_cont, _items, _pbase, _itco_)                          \
                                                                            \
STATEMENT_                                                                  \
(                                                                           \
    if ((_items) > (_cont)->_itco_##_limit && (_items) != 0)                \
    {                                                                       \
        size_t _size = sizeof(*(_cont)->_itco_);                            \
        int _ttl = _itco_total((_cont), _itco_);                            \
                                                                            \
        void **_tmp = (void*)&((_cont)->_itco_);                            \
        *_tmp = realloc(*_tmp, (_size *                                     \
                                (unsigned)((_cont)->_itco_##_base + (_items))));\
                                                                            \
        memset((_cont)->_itco_ + _ttl, 0,                                   \
               (_size * (unsigned)((_cont)->_itco_##_base + (_items) - _ttl)));\
                                                                            \
        (_cont)->_itco_##_limit = (_items);                                 \
    }                                                                       \
                                                                            \
    *(_pbase) = (_cont)->_itco_##_base;                                     \
                                                                            \
    (_cont)->_itco_##_base  += (_items);                                    \
    (_cont)->_itco_##_limit -= (_items);                                    \
```

```
)


#define _auxr_clear(_cont, _items, _itco_)                                  \
                                                                            \
STATEMENT_                                                                  \
(                                                                           \
    (_cont)->_itco_##_base  -= (_items);                                    \
    (_cont)->_itco_##_limit += (_items);                                    \
)


#define _auxr_free(_cont, _itco_)                                           \
                                                                            \
STATEMENT_                                                                  \
(                                                                           \
    if (_itco_total((_cont), _itco_) != 0)                                  \
        _safe_free ((_cont)->_itco_);                                       \
                                                                            \
    (_cont)->_itco_##_base  = 0;                                            \
    (_cont)->_itco_##_limit = 0;                                            \
)



/* node / block management */


#define _node_alloc(_pnode, _cont)                                          \
                                                                            \
STATEMENT_                                                                  \
(                                                                           \
    if ((_cont)->avsp == NULL)                                              \
    {                                                                       \
        if ((_cont)->vcnt == 0)                                             \
        {                                                                   \
            (_cont)->pblock = (_cont)->pool;                                \
                                                                            \
            if ((_cont)->pool != NULL && (_cont)->last != 0)                \
                (_cont)->pool = (_cont)->pool->bnxt;                        \
                                                                            \
            if ((_cont)->last == 0)                                         \
                (_cont)->vcnt = ((_cont)->last  = (_cont)->start);          \
            else                                                            \
                (_cont)->vcnt = ((_cont)->last  < (_cont)->thrsh) ?         \
                                ((_cont)->last *= (_cont)->ratio) :         \
                                ((_cont)->last  = (_cont)->thrsh);          \
                                                                            \
            if ((_cont)->pool == NULL)                                      \
            {                                                               \
```

```
                _safe_alloc((_cont)->pool, (sizeof(*(_cont)->pblock)) +        \
                                           (sizeof(*(_cont)->pblock->nodes)) * \
                                           (size_t)((_cont)->vcnt));           \
                                                                               \
                (_cont)->pool->bprv = (_cont)->pblock;                         \
                (_cont)->pool->ncnt = (_cont)->vcnt;                           \
                                                                               \
                (_cont)->pool->bnxt = NULL;                                    \
                if ((_cont)->pool->bprv != NULL)                               \
                    (_cont)->pool->bprv->bnxt = (_cont)->pool;                 \
            }                                                                  \
        }                                                                      \
                                                                               \
        (_pnode) = &((_cont)->pool->nodes[--(_cont)->vcnt]);                   \
    }                                                                          \
    else                                                                       \
    {                                                                          \
        (_pnode)      = (_cont)->avsp;                                         \
        (_cont)->avsp = (_cont)->avsp->lnk[0];                                 \
    }                                                                          \
)


#define _node_clear(_pnode, _cont)                                             \
                                                                               \
STATEMENT_                                                                     \
(                                                                              \
    (_pnode)->lnk[0] = (_cont)->avsp;                                          \
    (_cont)->avsp    = (_pnode);                                               \
)


#define _block_free(_cont)                                                     \
                                                                               \
STATEMENT_                                                                     \
(                                                                              \
    while ((_cont)->pool != NULL)                                              \
    {                                                                          \
        (_cont)->pblock = (_cont)->pool;                                       \
        (_cont)->pool   = (_cont)->pool->bprv;                                 \
        _safe_free((_cont)->pblock);                                           \
    }                                                                          \
)



#endif
```

1.2.4 base/snym.h

```
#ifndef OPENGC3_BASE_SNYM_H
#define OPENGC3_BASE_SNYM_H


/* dereference */

// ccdll
#define DREF(_iter)         ((_iter)->curr.node->val)
#define DREF_PREV(_iter)    ((_iter)->curr.node->PRV->val)
#define DREF_NEXT(_iter)    ((_iter)->curr.node->NXT->val)

// ccsll
#define SREF(_iter)         ((_iter)->curr.node->val)
#define SREF_NEXT(_iter)    ((_iter)->curr.node->NXT->val)

// ccxll
#define XREF(_iter)         ((_iter)->curr.node->val)
#define XREF_PREV(_iter)    ((_iter)->prev.node->val)
#define XREF_NEXT(_iter)    ((_iter)->next.node->val)

// cc[dsx]ll
#define LREF(_iter)         ((_iter)->curr.node->val)

// ccgbt
#define GREF(_iter)         ((_iter)->curr.node->val)
#define GREF_PARENT(_iter)  ((_iter)->curr.node->PRN->val)
#define GREF_LEFT(_iter)    ((_iter)->curr.node->LFT->val)
#define GREF_RIGHT(_iter)   ((_iter)->curr.node->RGH->val)


/* abbreviation */

// ccdll
#define DLEQ       ccdll_comp_leq
#define DLEQ_PREV  ccdll_comp_leq_prev
#define DLEQ_NEXT  ccdll_comp_leq_next
#define DGEQ       ccdll_comp_geq
#define DGEQ_PREV  ccdll_comp_geq_prev
#define DGEQ_NEXT  ccdll_comp_geq_next

// ccsll
#define SLEQ       ccsll_comp_leq
#define SLEQ_NEXT  ccsll_comp_leq_next
#define SGEQ       ccsll_comp_geq
#define SGEQ_NEXT  ccsll_comp_geq_next

// ccxll
#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
```