Data_Struct_In_C By JackJin 2014

v0.1

Doxygen 1.8.6

2014 6 10:18:11

Contents

Chapter 1

DataStrut in C JackJin

****

1. Algorithms in C, Third Edition

**

ToDoList

2014-03-05

~~1.queuestack~~

1.queuestack

Chapter 2

****

1. ——
2.

****

1.
2. C
3.

******

```
$ echo "Welcome to Bash" ***2***
```

Chapter 3

3.1

:

Chapter 4

4.1

:

Chapter 5

5.1 list

- Node **head**

5.1.1

d_list.h 32 .

:

- list/d_list.h
- list/s_list.h
- list/sc_list.h
- stack/stack_list.c

5.2 node

- struct node * **prev**
- Ele **ele**
- struct node * **next**
- PNode **next**

5.2.1

d_list.h 22 .

:

- list/d_list.h
- list/s_list.h
- list/sc_list.h
- queue/queue_adt/queue.h
- queue/queue_list.c
- stack/stack_list.c

5.3 point

- float **x**
- float **y**

5.3.1

point.h 4 .

:

- array/include/point.h

5.4 queue

- PNode **head**
- PNode **tail**
- size_t **counter**

5.4.1

queue.h 29 .

:

- queue/queue_adt/queue.h

5.5 vector

- PEle **array**
- size_t **counter**
- size_t **max**

5.5.1

vector.h 8 .

:

- vector/vector.h

Chapter 6

6.1 array/src/1_eratossthenes.c

1-N

```
#include <stdio.h>
```

- #define **N** 100

- int **main** (void)

6.1.1

1-N

Jack Jin - gjinjian@gmail.com

v0.1

2014-01-17

1_eratossthenes.c .

6.2 array/src/2_bernoulli_trial.c

f[cnt]——cnt N / M

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

- int **heads** ()
- int **main** (int argc, char ∗argv[])

6.2.1

f[cnt]——cnt N / M

Jack Jin - gjinjian@gmail.com

v0.1

2014-01-18

2_bernoulli_trial.c .

6.3 array/src/3_near_point.c

N,N!2d

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "Point.h"
```

- float **randFloat** ()
- int **main** (int argc, char ∗argv[])

6.3.1

N,N!2d

Jack Jin - gjinjian@gmail.com

v0.1

2014-01-19

3_near_point.c .

6.4 element.h

- typedef int **Ele**
- typedef int * **PEle**
- typedef unsigned int **size_t**

6.4.1

Jack Jin - gjinjian@gmail.com

v0.1

2014-01-19

element.h .

6.5 list/d_list.c

index 0——size-1

```
#include "d_list.h"
#include <malloc.h>
#include <assert.h>
#include <stdio.h>
```

- PList create (void)
- void destory (PList pl)

  pl
- int push_front (PList pl, PEle pe)

  plpe
- int push_back (PList pl, PEle pe)

  plpe
- int insert (PList pl, size_t index, PEle pe)

  plindexpe :1:index: 2index:
- int remove_front (PList pl)

  pl
- int remove_back (PList pl)
- int mremove (PList pl, size_t index)

  index
- int set_front (PList pl, PEle pe)

- int set_back (PList pl, PEle pe)
- int set (PList pl, size_t index, PEle pe)

  index
- PNode get_front (PList pl)
- PNode get_back (PList pl)
- PNode get (PList pl, size_t index)

  index
- size_t size (PList pl)
- int show (PList pl)

6.5.1

index 0——size-1

Jack Jin - gjinjian@gmail.com

v0.1

2014-01-22

d_list.c .

6.5.2

6.5.2.1 PList create (void)

,NULL

d_list.c 23 .

```
24 {
25     PList pl = (PList)malloc(sizeof(List));
26     if(pl == NULL)
27         return NULL;
28     pl->head.prev = pl->head.next = NULL;
29 
30     return pl;
31 }
```

6.5.2.2 void destory (PList *pl*)

pl

PList	

d_list.c 53 .

```
54 {
55     assert(pl != NULL);
56     destory_node(pl->head.next);
57     free(pl);
58     pl = NULL;
59 }
```

6.5.2.3 PNode get (PList *pl*, size_t *index*)

index

PList	
index	

10

d_list.c 378 .

```
379 {
380     if(pl == NULL || index > size(pl))
381         return NULL;
382     PNode pn = pl->head.next;
383     while(index-- > 0)
384         pn = pn->next;
385
386     return pn;
387 }
```

6.5.2.4 PNode get_back (PList *pl*)

PList	

10

d_list.c 364 .

```
365 {
366     return get(pl, size(pl)-1);
367 }
```

6.5.2.5 PNode get_front (PList *pl*)

—

PList	

10

d_list.c 351 .

```
352 {
353     return get(pl, 0);
354 }
```

6.5.2.6 int insert (PList *pl*, size_t *index*, PEle *pe*)

plindexpe :1:index: 2index:

head index[0——(size-1)]

PList	
index	
PEle	

10

d_list.c 179 .

```
180 {
181     if(pl == NULL || pe == NULL || index > size(pl))
182         return 0;
183 #if 0
184     PNode pre = get_prev(pl->head, index);
185     if(pre == NULL) return 0;
186 
187     PNode pn = (PNode)malloc(sizeof(Node));
188     if(pn == NULL)  return 0;
189     pn->ele = *pe;
190     pn->prev = pre;
191     pn->next = pre->next;
192 
193     if(pre->next != NULL){
194         pre->next = pn;
195         pre->next->prev = pn;
196     }
197 
198     return 1;
199 
200 #endif
201     //2
202     PNode pn = get(pl, index);
203     if(pn == NULL)
204         return 0;
205 
206     PNode node = (PNode)malloc(sizeof(Node));
207     if(node == NULL)
208         return 0;
209 
210     node->prev = pn->prev;
211     node->ele  = *pe;
212     node->next = pn;
213 
214     if(pn->prev != NULL)
215         pn->prev->next = node;
216 
217     pn->prev = node;
218     return 1;
219 }
```

6.5.2.7 int mremove (PList *pl*, size_t *index*)

index

PList	
index	

10

d_list.c 258 .

```
259 {
260     if(pl == NULL || index > size(pl))
261         return 0;
262 #if 0
263     PNode pn = get_prev(pl->head);
264     if(pn == NULL || pn->next == NULL)   return 0;
265
266     PNode tmp = pn->next;
267
268     pn->next = tmp->next;
269     if(tmp->next == NULL)  return 0;
270     tmp->next->prev = pn;
271
272     free(tmp);
273     tmp = NULL;
274 #endif
275
276     // find
277     PNode pn = get(pl, index);
278     if(pn == NULL)  return 0;
279     // swap
280     if(pn->next != NULL){
281         pn->prev->next = pn->next;
282         pn->next->prev = pn->prev;
283     }else{
284         pn->prev->next = NULL;
285     }
286     // free
287     free(pn);
288     pn = NULL;
289     return 1;
290 }
```

6.5.2.8 int push_back (PList *pl*, PEle *pe*)

plpe

PList	
PEle	

10

null

d_list.c 121 .

```
122 {
123     if(pl == NULL || pe == NULL)
124         return 0;
125
126     //PNode end = tail(pl->head.next);
127     PNode end = tail(&pl->head);
128
129     PNode pn = (PNode)malloc(sizeof(Node));
130     if(pn == NULL)  return 0;
131     pn->prev = end;
132     pn->ele  = *pe;
133     pn->next = NULL;
134
```

```
135     end->next = pn;
136
137     return 1;
138 }
```

6.5.2.9 int push_front (PList *pl*, PEle *pe*)

plpe

PList	
PEle	

10

d_list.c 71 .

```
72 {
73     if(pl == NULL || pe == NULL)
74         return 0;
75
76     PNode pn = (PNode)malloc(sizeof(Node));
77     if(pn == NULL)
78         return 0;
79     pn->prev = pn->next = NULL;
80     // head ---->[] ----> 1st
81     // []
82     pn->prev = &pl->head;
83     pn->ele = *pe;
84     pn->next = pl->head.next;
85     // 1st
86     if(pn->next != NULL)
87         pn->next->prev = pn;
88     // head
89     pl->head.next = pn;
90     return 1;
91 }
```

6.5.2.10 int remove_back (PList *pl*)

PList	

10

d_list.c 244 .

```
245 {
246     return mremove(pl, size(pl) -1);
247 }
```

6.5.2.11 int remove_front (PList *pl*)

pl

PList	

10

d_list.c 230 .

```
231 {
232     return mremove(pl, 0);
233 }
```

6.5.2.12 int set (PList *pl*, size_t *index*, PEle *pe*)

index

PList	
index	
PEle	

10

d_list.c 330 .

```
331 {
332     if(pl == NULL || pe == NULL || index > size(pl))
333         return 0;
334     PNode pn = get(pl, index);
335     if(pn == NULL)  return 0;
336     pn->ele = *pe;
337     return 1;
338 }
```

6.5.2.13 int set_back (PList *pl*, PEle *pe*)

PList	
PEle	

10

d_list.c 315 .

```
316 {
317     return set(pl, size(pl) -1, pe);
318 }
```

6.5.2.14 int set_front (PList *pl*, PEle *pe*)

PList	
PEle	

10

d_list.c 301 .

```
302 {
303     return set(pl, 0, pe);
304 }
```

6.5.2.15 int show (PList *pl*)

PList	

10

d_list.c 419 .

```
420 {
421     if(pl == NULL) return 0;
422     printf("Index\tEle Size:%d \n", size(pl));
423     int i = 0;
424     PNode pn = pl->head.next;
425 
426     while(i != size(pl)){
427         printf("%-4d %-4d\n", i++, pn->ele);
428         pn = pn->next;
429     }
430     return 1;
431 }
```

6.5.2.16 size_t size (PList *pl*)

PList	

10

d_list.c 397 .

```
398 {
399     if(pl == NULL)
400         return 0;
401 
402     int cnt = 0;
403     PNode pn = pl->head.next;
404     while(pn != NULL){
405         pn = pn->next;
406         cnt++;
407     }
408     return cnt;
409 }
```

6.6 list/d_list.h

```
#include "element.h"
```

- struct node
- struct list

- typedef struct node **Node**
- typedef struct node * **PNode**
- typedef struct list **List**
- typedef struct list * **PList**

- PList create (void)

- void destory (PList)
  pl
- int push_front (PList, PEle)
  plpe
- int push_back (PList, PEle)
  plpe
- int insert (PList, size_t index, PEle)
  plindexpe :1:index: 2index:
- int remove_front (PList)
  pl
- int remove_back (PList)

- int mremove (PList, size_t index)
  index
- int set_front (PList, PEle)

- int set_back (PList, PEle)

- int set (PList, size_t index, PEle)
  index
- PNode get_front (PList)

- PNode get_back (PList)

- PNode get (PList, size_t index)
  index
- size_t size (PList)

- int show (PList)

6.6.1

Jack Jin - gjinjian@gmail.com

v0.1

2014-01-22

d_list.h .

6.6.2

6.6.2.1 PList create (void)

,NULL

PVector

d_list.c 23 .

```
24 {
25     PList pl = (PList)malloc(sizeof(List));
26     if(pl == NULL)
27         return NULL;
28     pl->head.prev = pl->head.next = NULL;
29 
30     return pl;
31 }
```

6.6.2.2 void destory (PList *pl*)

pl

PList	pl
PList	

d_list.c 53 .

```
54 {
55     assert(pl != NULL);
56     destory_node(pl->head.next);
57     free(pl);
58     pl = NULL;
59 }
```

6.6.2.3 PNode get (PList *pl*, size_t *index*)

index

PList	
index	

10

d_list.c 378 .

```
379 {
380     if(pl == NULL || index > size(pl))
381         return NULL;
382     PNode pn = pl->head.next;
383     while(index-- > 0)
384         pn = pn->next;
385
386     return pn;
387 }
```

6.6.2.4 PNode get_back (PList *pl*)

PList	

10

d_list.c 364 .

```
365 {
366     return get(pl, size(pl)-1);
367 }
```

6.6.2.5 PNode get_front (PList *pl*)

PList	

10

d_list.c 351 .

```
352 {
353     return get(pl, 0);
354 }
```

6.6.2.6 int insert (PList *pl*, size_t *index*, PEle *pe*)

plindexpe :1:index: 2index:

head index[0——(size-1)]

PList	
index	
PEle	

10

d_list.c 179 .

```
180 {
181     if(pl == NULL || pe == NULL || index > size(pl))
182         return 0;
183 #if 0
184     PNode pre = get_prev(pl->head, index);
185     if(pre == NULL) return 0;
186 
187     PNode pn = (PNode)malloc(sizeof(Node));
188     if(pn == NULL)  return 0;
189     pn->ele = *pe;
190     pn->prev = pre;
191     pn->next = pre->next;
192 
193     if(pre->next != NULL){
194         pre->next = pn;
195         pre->next->prev = pn;
196     }
197 
198     return 1;
199 
200 #endif
201     //2
202     PNode pn = get(pl, index);
203     if(pn == NULL)
204         return 0;
205 
206     PNode node = (PNode)malloc(sizeof(Node));
207     if(node == NULL)
208         return 0;
209 
210     node->prev = pn->prev;
211     node->ele  = *pe;
212     node->next = pn;
213 
214     if(pn->prev != NULL)
215         pn->prev->next = node;
216 
217     pn->prev = node;
218     return 1;
219 }
```

6.6.2.7 int mremove (PList *pl*, size_t *index*)

index

PList	
index	

10

d_list.c 258 .

```
259 {
260     if(pl == NULL || index > size(pl))
261         return 0;
262 #if 0
263     PNode pn = get_prev(pl->head);
264     if(pn == NULL || pn->next == NULL)   return 0;
```

```
265
266     PNode tmp = pn->next;
267
268     pn->next = tmp->next;
269     if(tmp->next == NULL)  return 0;
270     tmp->next->prev = pn;
271
272     free(tmp);
273     tmp = NULL;
274 #endif
275
276     // find
277     PNode pn = get(pl, index);
278     if(pn == NULL)  return 0;
279     // swap
280     if(pn->next != NULL){
281         pn->prev->next = pn->next;
282         pn->next->prev = pn->prev;
283     }else{
284         pn->prev->next = NULL;
285     }
286     // free
287     free(pn);
288     pn = NULL;
289     return 1;
290 }
```

6.6.2.8 int push_back (PList *pl*, PEle *pe*)

plpe

PList	
PEle	

10

null

d_list.c 121 .

```
122 {
123     if(pl == NULL || pe == NULL)
124         return 0;
125
126     //PNode end = tail(pl->head.next);
127     PNode end = tail(&pl->head);
128
129     PNode pn = (PNode)malloc(sizeof(Node));
130     if(pn == NULL)  return 0;
131     pn->prev = end;
132     pn->ele  = *pe;
133     pn->next = NULL;
134
135     end->next = pn;
136
137     return 1;
138 }
```

6.6.2.9 int push_front (PList *pl*, PEle *pe*)

plpe

—

PList	
PEle	

10

d_list.c 71 .

```
72 {
73     if(pl == NULL || pe == NULL)
74         return 0;
75
76     PNode pn = (PNode)malloc(sizeof(Node));
77     if(pn == NULL)
78         return 0;
79     pn->prev = pn->next = NULL;
80     // head ----->[] -----> 1st
81     // []
82     pn->prev = &pl->head;
83     pn->ele = *pe;
84     pn->next = pl->head.next;
85     // 1st
86     if(pn->next != NULL)
87         pn->next->prev = pn;
88     // head
89     pl->head.next = pn;
90     return 1;
91 }
```

6.6.2.10 int remove_back (PList *pl*)

PList	

10

d_list.c 244 .

```
245 {
246     return mremove(pl, size(pl) -1);
247 }
```

6.6.2.11 int remove_front (PList *pl*)

pl

PList	

10

d_list.c 230 .

```
231 {
232     return mremove(pl, 0);
233 }
```

6.6.2.12 int set (PList *pl*, size_t *index*, PEle *pe*)

index

PList	
index	
PEle	

10

d_list.c 330 .

```
331 {
332     if(pl == NULL || pe == NULL || index > size(pl))
333         return 0;
334     PNode pn = get(pl, index);
335     if(pn == NULL)  return 0;
336     pn->ele = *pe;
337     return 1;
338 }
```

6.6.2.13 int set_back (PList *pl*, PEle *pe*)

PList	
PEle	

10

d_list.c 315 .

```
316 {
317     return set(pl, size(pl) -1, pe);
318 }
```

6.6.2.14 int set_front (PList *pl*, PEle *pe*)

PList	
PEle	

10

d_list.c 301 .

```
302 {
303     return set(pl, 0, pe);
304 }
```

6.6.2.15 int show (PList *pl*)

PList	

10

d_list.c 419 .

```
420 {
421     if(pl == NULL) return 0;
422     printf("Index\tEle Size:%d \n", size(pl));
423     int i = 0;
424     PNode pn = pl->head.next;
425 
426     while(i != size(pl)){
427         printf("%-4d %-4d\n", i++, pn->ele);
428         pn = pn->next;
429     }
430     return 1;
431 }
```

6.6.2.16 size_t size (PList *pl*)

PList	

10

d_list.c 397 .

```
398 {
399     if(pl == NULL)
400         return 0;
401 
402     int cnt = 0;
403     PNode pn = pl->head.next;
404     while(pn != NULL){
405         pn = pn->next;
406         cnt++;
407     }
408     return cnt;
409 }
```

6.7 list/s_list.h

```
#include "element.h"
```

- struct node
- struct list

- typedef unsigned int **size_t**
- typedef struct node **Node**
- typedef struct node * **PNode**
- typedef struct list **List**
- typedef struct list * **PList**

- PList create (void)
- void destory (PList)

  pl
- int push_front (PList, PEle)

  plpe
- int push_back (PList, PEle)

  plpe
- int insert (PList, size_t index, PEle)

  plindexpe :1:index: 2index:
- int remove_front (PList)

  pl
- int remove_back (PList)
- int mremove (PList, size_t index)

  index
- int set_front (PList, PEle)
- int set_back (PList, PEle)
- int set (PList, size_t index, PEle)

  index
- PNode get_front (PList)
- PNode get_back (PList)
- PNode get (PList, size_t index)

  index
- size_t size (PList)
- int show (PList)

6.7.1

Jack Jin - gjinjian@gmail.com

v0.1

2014-01-22

s_list.h .

6.7.2

6.7.2.1 PList create (void)

,NULL

PVector

d_list.c 23 .

```
24 {
25     PList pl = (PList)malloc(sizeof(List));
26     if(pl == NULL)
27         return NULL;
28     pl->head.prev = pl->head.next = NULL;
29
30     return pl;
31 }
```

6.7.2.2 void destory (PList *pl*)

pl

PList	pl
PList	

d_list.c 53 .

```
54 {
55     assert(pl != NULL);
56     destory_node(pl->head.next);
57     free(pl);
58     pl = NULL;
59 }
```

6.7.2.3 PNode get (PList *pl*, size_t *index*)

index

PList	
index	

10

d_list.c 378 .

```
379 {
380     if(pl == NULL || index > size(pl))
381         return NULL;
382     PNode pn = pl->head.next;
383     while(index-- > 0)
384         pn = pn->next;
385
386     return pn;
387 }
```

6.7.2.4 PNode get_back (PList *pl*)

PList	

10

d_list.c 364 .

```
365 {
366     return get(pl, size(pl)-1);
367 }
```

6.7.2.5 PNode get_front (PList *pl*)

PList	

10

d_list.c 351 .

```
352 {
353     return get(pl, 0);
354 }
```

6.7.2.6 int insert (PList *pl*, size_t *index*, PEle *pe*)

plindexpe :1:index: 2index:

head index[0——(size-1)]

PList	
index	
PEle	

10

d_list.c 179 .

```
180 {
181     if(pl == NULL || pe == NULL || index > size(pl))
182         return 0;
183 #if 0
184     PNode pre = get_prev(pl->head, index);
185     if(pre == NULL) return 0;
186
187     PNode pn = (PNode)malloc(sizeof(Node));
188     if(pn == NULL)  return 0;
189     pn->ele = *pe;
190     pn->prev = pre;
191     pn->next = pre->next;
192
193     if(pre->next != NULL){
194         pre->next = pn;
195         pre->next->prev = pn;
196     }
197
```

```
198     return 1;
199
200 #endif
201     //2
202     PNode pn = get(pl, index);
203     if(pn == NULL)
204         return 0;
205
206     PNode node = (PNode)malloc(sizeof(Node));
207     if(node == NULL)
208         return 0;
209
210     node->prev = pn->prev;
211     node->ele  = *pe;
212     node->next = pn;
213
214     if(pn->prev != NULL)
215         pn->prev->next = node;
216
217     pn->prev = node;
218     return 1;
219 }
```

6.7.2.7 int mremove (PList *pl*, size_t *index*)

index

PList	
index	

10

d_list.c 258 .

```
259 {
260     if(pl == NULL || index > size(pl))
261         return 0;
262 #if 0
263     PNode pn = get_prev(pl->head);
264     if(pn == NULL || pn->next == NULL)    return 0;
265
266     PNode tmp = pn->next;
267
268     pn->next = tmp->next;
269     if(tmp->next == NULL)   return 0;
270     tmp->next->prev = pn;
271
272     free(tmp);
273     tmp = NULL;
274 #endif
275
276     // find
277     PNode pn = get(pl, index);
278     if(pn == NULL)   return 0;
279     // swap
280     if(pn->next != NULL){
281         pn->prev->next = pn->next;
282         pn->next->prev = pn->prev;
283     }else{
284         pn->prev->next = NULL;
285     }
286     // free
287     free(pn);
288     pn = NULL;
289     return 1;
290 }
```

6.7.2.8 int push_back (PList *pl*, PEle *pe*)

plpe

PList	
PEle	

10

null

null

null

d_list.c 121 .

```
122 {
123     if(pl == NULL || pe == NULL)
124         return 0;
125
126     //PNode end = tail(pl->head.next);
127     PNode end = tail(&pl->head);
128
129     PNode pn = (PNode)malloc(sizeof(Node));
130     if(pn == NULL)  return 0;
131     pn->prev = end;
132     pn->ele  = *pe;
133     pn->next = NULL;
134
135     end->next = pn;
136
137     return 1;
138 }
```

6.7.2.9 int push_front (PList *pl*, PEle *pe*)

plpe

PList	
PEle	

10

d_list.c 71 .

```
72 {
73     if(pl == NULL || pe == NULL)
74         return 0;
75
76     PNode pn = (PNode)malloc(sizeof(Node));
77     if(pn == NULL)
78         return 0;
79     pn->prev = pn->next = NULL;
80     // head ---->[] ----> 1st
81     // []
82     pn->prev = &pl->head;
83     pn->ele = *pe;
84     pn->next = pl->head.next;
85     // 1st
86     if(pn->next != NULL)
87         pn->next->prev = pn;
88     // head
89     pl->head.next = pn;
90     return 1;
91 }
```

6.7.2.10 int remove_back (PList *pl*)

PList	

10

d_list.c 244 .

```
245 {
246     return mremove(pl, size(pl) -1);
247 }
```

6.7.2.11 int remove_front (PList *pl*)

pl

PList	

10

d_list.c 230 .

```
231 {
232     return mremove(pl, 0);
233 }
```

6.7.2.12 int set (PList *pl*, size_t *index*, PEle *pe*)

index

PList	
index	
PEle	

10

d_list.c 330 .

```
331 {
332     if(pl == NULL || pe == NULL || index > size(pl))
333         return 0;
334     PNode pn = get(pl, index);
335     if(pn == NULL)  return 0;
336     pn->ele = *pe;
337     return 1;
338 }
```

6.7.2.13 int set_back (PList *pl*, PEle *pe*)

PList	
PEle	

10

d_list.c 315 .

```
316 {
317     return set(pl, size(pl) -1, pe);
318 }
```

6.7.2.14 int set_front (PList *pl*, PEle *pe*)

PList	
PEle	

10

d_list.c 301 .

```
302 {
303     return set(pl, 0, pe);
304 }
```

6.7.2.15 int show (PList *pl*)

PList	

10

d_list.c 419 .

```
420 {
421     if(pl == NULL) return 0;
422     printf("Index\tEle Size:%d \n", size(pl));
423     int i = 0;
424     PNode pn = pl->head.next;
425 
426     while(i != size(pl)){
427         printf("%-4d %-4d\n", i++, pn->ele);
428         pn = pn->next;
429     }
430     return 1;
431 }
```

6.7.2.16 size_t size (PList *pl*)

PList	

10

d_list.c 397 .

```
398 {
399     if(pl == NULL)
400         return 0;
401
402     int cnt = 0;
403     PNode pn = pl->head.next;
404     while(pn != NULL){
405         pn = pn->next;
406         cnt++;
407     }
408     return cnt;
409 }
```

6.8 list/sc_list.h

```
#include "element.h"
```

- struct node
- struct list

- typedef struct node **Node**
- typedef struct node * **PNode**
- typedef struct list **List**
- typedef struct list * **PList**

- PList create (void)

- void destory (PList)

  pl
- int push_front (PList, PEle)

  plpe
- int push_back (PList, PEle)

  plpe
- int insert (PList, size_t index, PEle)

  plindexpe :1:index: 2index:
- int remove_front (PList)

  pl
- int remove_back (PList)

- int mremove (PList, size_t index)
  index
- int set_front (PList, PEle)
- int set_back (PList, PEle)
- int set (PList, size_t index, PEle)
  index
- PNode get_front (PList)
- PNode get_back (PList)
- PNode get (PList, size_t index)
  index
- size_t size (PList)
- int show (PList)

6.8.1

Jack Jin - gjinjian@gmail.com

v0.1

2014-02-14

sc_list.h .

6.8.2

6.8.2.1 PList create (void)

,NULL

PVector

d_list.c 23 .

```
24 {
25     PList pl = (PList)malloc(sizeof(List));
26     if(pl == NULL)
27         return NULL;
28     pl->head.prev = pl->head.next = NULL;
29
30     return pl;
31 }
```

6.8.2.2 void destory (PList *pl*)

pl

PList	pl
PList	

d_list.c 53 .

```
54 {
55     assert(pl != NULL);
56     destory_node(pl->head.next);
57     free(pl);
58     pl = NULL;
59 }
```

6.8.2.3 PNode get (PList *pl*, size_t *index*)

index

PList	
index	

10

d_list.c 378 .

```
379 {
380     if(pl == NULL || index > size(pl))
381         return NULL;
382     PNode pn = pl->head.next;
383     while(index-- > 0)
384         pn = pn->next;
385
386     return pn;
387 }
```

6.8.2.4 PNode get_back (PList *pl*)

PList	

10

d_list.c 364 .

```
365 {
366     return get(pl, size(pl)-1);
367 }
```

6.8.2.5 PNode get_front (PList *pl*)

PList	

10

d_list.c 351 .

```
352 {
353     return get(pl, 0);
354 }
```

6.8.2.6 int insert (PList *pl*, size_t *index*, PEle *pe*)

plindexpe :1:index: 2index:

head index[0——(size-1)]

PList	
index	
PEle	

10

d_list.c 179 .

```
180 {
181     if(pl == NULL || pe == NULL || index > size(pl))
182         return 0;
183 #if 0
184     PNode pre = get_prev(pl->head, index);
185     if(pre == NULL) return 0;
186 
187     PNode pn = (PNode)malloc(sizeof(Node));
188     if(pn == NULL)  return 0;
189     pn->ele = *pe;
190     pn->prev = pre;
191     pn->next = pre->next;
192 
193     if(pre->next != NULL){
194         pre->next = pn;
195         pre->next->prev = pn;
196     }
197 
198     return 1;
199 
200 #endif
201     //2
202     PNode pn = get(pl, index);
203     if(pn == NULL)
204         return 0;
205 
206     PNode node = (PNode)malloc(sizeof(Node));
207     if(node == NULL)
208         return 0;
209 
210     node->prev = pn->prev;
211     node->ele  = *pe;
212     node->next = pn;
213 
214     if(pn->prev != NULL)
215         pn->prev->next = node;
216 
217     pn->prev = node;
218     return 1;
219 }
```

6.8.2.7 int mremove (PList *pl,* size_t *index*)

index

PList	
index	

10

d_list.c 258 .

```
259 {
260     if(pl == NULL || index > size(pl))
261         return 0;
262 #if 0
263     PNode pn = get_prev(pl->head);
264     if(pn == NULL || pn->next == NULL)    return 0;
265 
266     PNode tmp = pn->next;
267 
268     pn->next = tmp->next;
269     if(tmp->next == NULL)   return 0;
270     tmp->next->prev = pn;
271 
272     free(tmp);
273     tmp = NULL;
274 #endif
275 
276     // find
277     PNode pn = get(pl, index);
278     if(pn == NULL)   return 0;
279     // swap
280     if(pn->next != NULL){
281         pn->prev->next = pn->next;
282         pn->next->prev = pn->prev;
283     }else{
284         pn->prev->next = NULL;
285     }
286     // free
287     free(pn);
288     pn = NULL;
289     return 1;
290 }
```

6.8.2.8 int push_back (PList *pl*, PEle *pe*)

plpe

PList	
PEle	

10

null

null

null

d_list.c 121 .

```
122 {
123     if(pl == NULL || pe == NULL)
124         return 0;
125 
126     //PNode end = tail(pl->head.next);
127     PNode end = tail(&pl->head);
128 
129     PNode pn = (PNode)malloc(sizeof(Node));
```

```
130     if(pn == NULL)  return 0;
131     pn->prev = end;
132     pn->ele  = *pe;
133     pn->next = NULL;
134
135     end->next = pn;
136
137     return 1;
138 }
```

6.8.2.9 int push_front (PList *pl*, PEle *pe*)

plpe

PList	
PEle	

10

d_list.c 71 .

```
72 {
73     if(pl == NULL || pe == NULL)
74         return 0;
75
76     PNode pn = (PNode)malloc(sizeof(Node));
77     if(pn == NULL)
78         return 0;
79     pn->prev = pn->next = NULL;
80     // head ---->[] ----> lst
81     // []
82     pn->prev = &pl->head;
83     pn->ele = *pe;
84     pn->next = pl->head.next;
85     // lst
86     if(pn->next != NULL)
87         pn->next->prev = pn;
88     // head
89     pl->head.next = pn;
90     return 1;
91 }
```

6.8.2.10 int remove_back (PList *pl*)

PList	

10

d_list.c 244 .

```
245 {
246     return mremove(pl, size(pl) -1);
247 }
```

6.8.2.11 int remove_front (PList *pl*)

pl

PList	

10

d_list.c 230 .

```
231 {
232     return mremove(pl, 0);
233 }
```

6.8.2.12 int set (PList *pl*, size_t *index*, PEle *pe*)

index

PList	
index	
PEle	

10

d_list.c 330 .

```
331 {
332     if(pl == NULL || pe == NULL || index > size(pl))
333         return 0;
334     PNode pn = get(pl, index);
335     if(pn == NULL)  return 0;
336     pn->ele = *pe;
337     return 1;
338 }
```

6.8.2.13 int set_back (PList *pl*, PEle *pe*)

PList	
PEle	

10

d_list.c 315 .

```
316 {
317     return set(pl, size(pl) -1, pe);
318 }
```

6.8.2.14 int set_front (PList *pl*, PEle *pe*)

PList	
PEle	

10

d_list.c 301 .

```
302 {
303     return set(pl, 0, pe);
304 }
```

6.8.2.15 int show (PList *pl*)

PList	

10

d_list.c 419 .

```
420 {
421     if(pl == NULL) return 0;
422     printf("Index\tEle Size:%d \n", size(pl));
423     int i = 0;
424     PNode pn = pl->head.next;
425 
426     while(i != size(pl)){
427         printf("%-4d %-4d\n", i++, pn->ele);
428         pn = pn->next;
429     }
430     return 1;
431 }
```

6.8.2.16 size_t size (PList *pl*)

PList	

10

d_list.c 397 .

```
398 {
399     if(pl == NULL)
400         return 0;
401 
402     int cnt = 0;
403     PNode pn = pl->head.next;
404     while(pn != NULL){
405         pn = pn->next;
406         cnt++;
407     }
408     return cnt;
409 }
```

6.9 list/sc_list_test.c

```
#include "sc_list.h"
#include <stdio.h>
#include <assert.h>
```

- int **main** (void)

6.9.1

Jack Jin - gjinjian@gmail.com

v0.1

2014-02-16 2014-02-16 19:33

sc_list_test.c .

6.10 queue/queue_list.c

```
#include "queue.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <assert.h>
```

- struct node

- typedef struct node **Node**
- typedef struct node * **PNode**

- void init (int max)

- int enqueue (Ele ele)

  ele
- Ele dequeue (void)

- Ele cpqueue (void)
- bool is_empty (void)
- bool is_full (void)

  , *false*
- size_t size (void)

6.10.1

Jack Jin - gjinjian@gmail.com

V0.1

2014-03-03

queue_list.c .

6.10.2

6.10.2.1 Ele cpqueue (void)

queue_list.c 88 .

```
89 {
90     if(is_empty())  exit(1);
91     return head->ele;
92 }
```

6.10.2.2 Ele dequeue (void)

queue_list.c 70 .

```
71 {
72     if(is_empty())  exit(1);
73     Ele ele = head->ele;
74     PNode tmp = head->next;
75     free(head);
76     head = tmp;
77     counter--;
78
79     return ele;
80 }
```

6.10.2.3 int enqueue (Ele *ele*)

ele

Ele	

10

queue_list.c 48 .

```
49 {
50     if(is_empty()){
51         tail = NEW(ele, head);
52         head = tail;
53         counter++;
54         return 1;
55      }
56
57     tail->next = NEW(ele, tail->next);
58     tail = tail->next;
59     counter++;
60
61     return 1;
62 }
```

6.10.2.4 void init (int *max*)

max	

queue_list.c 35 .

```
36 {
37     head = NULL;
38     tail = NULL;
39     counter = 0;
40 }
```

6.10.2.5 bool is_empty (void)

queue_list.c 101 .

```
102 {
103     if (head == NULL)
104         return true;
105     else
106         return false;
107 }
```

6.10.2.6 bool is_full (void)

, false

queue_list.c 115 .

```
116 {
117     return false;
118 }
```

6.10.2.7 size_t size (void)

queue_list.c 126 .

```
127 {
128     return counter;
129 }
```

6.11 stack/stack_list.c

```
#include <stdlib.h>
#include "element.h"
#include "stack.h"
```

- struct node
- struct list

- typedef struct node **Node**
- typedef struct node * **PNode**
- typedef struct list **List**
- typedef struct list * **PList**

- void init (size_t max)
  stack
- int push (Ele ele)
  ,counter+1
- Ele pop (void)
  counter-1
- Ele top (void)
  toppopcounter
- bool is_full (void)
  ,false
- bool is_empty (void)

- size_t size (void)

6.11.1

Jack Jin - gjinjian@gmail.com

v0.1

2014-03-02

stack_list.c .

6.11.2

6.11.2.1 void init (size_t *max*)

stack

max	

stack_list.c 54 .

```
55 {
56     pl = (PList)malloc(sizeof(List));
57     assert(pl != NULL);
58
59     pl->head.next = NULL;          //
60     //pl->head.next = &pl->head;   //
61     counter = 0;
62 }
```

6.11.2.2 bool is_empty (void)

stack_list.c 121 .

```
122 {
123     if(pl->head.next == NULL)
124         return true;
125     else
126         return false;
127 }
```

6.11.2.3 bool is_full (void)

,false

false

stack_list.c 111 .

```
112 {
113     return false;
114 }
```

6.11.2.4 Ele pop (void)

counter-1

stack_list.c 82 .

```
83 {
84     if(is_empty())  exit(1);
85
86     Ele ele = pl->head.next->ele;
87     PNode pn = pl->head.next->next;
88     free(pl->head.next);
89     pl->head.next = pn;
90
91     counter--;
92     return ele;
93 }
```

6.11.2.5 int push (Ele *ele*)

,counter+1

Ele	

10

stack_list.c 71 .

```
72 {
73      pl->head.next = NEW(ele, pl->head.next);
74      ++counter;
75 }
```

6.11.2.6 size_t size (void)

stack_list.c 135 .

```
136 {
137     return counter;
138 }
```

6.11.2.7 Ele top (void)

toppopcounter

counter

stack_list.c 101 .

```
102 {
103     return pop();
104 }
```

6.12 vector/vector.c

```
#include "vector.h"
#include <stdio.h>
#include <malloc.h>
#include <memory.h>
#include <assert.h>
```

- #define INIT_SIZE 10

- PVector create (void)

  vector
- void destory (PVector pv)

  vector
- int push_front (PVector pv, PEle pe)
- int push_back (PVector pv, PEle pe)
- int insert (PVector pv, size_t index, PEle pe)

  ——
- int remove_front (PVector pv)

  0
- int remove_back (PVector pv)
- int mremove (PVector pv, size_t index)

  index
- int set_front (PVector pv, PEle pe)

  PVector∗pe
- int set_back (PVector pv, PEle pe)

  PVector∗pe
- int set (PVector pv, size_t index, PEle pe)

  PVectorindex∗pe
- PEle get_front (PVector pv)
- PEle get_back (PVector pv)
- PEle get (PVector pv, size_t index)

  PVectorindex
- size_t size (PVector pv)

  PVector counter
- int **show** (PVector pv)

6.12.1

Jack Jin - gjinjian@gmail.com

v0.1

2014-01-19

vector.c .

6.12.2

6.12.2.1 #define INIT_SIZE 10

vector.c 21 .

6.12.3

6.12.3.1 PVector create (void)

vector

PVector

vector.c 29 .

```
30 {
31     PVector pv = (PVector)malloc(sizeof(Vector));
32     assert(pv != NULL);
33
34     pv->array = calloc(INIT_SIZE, sizeof(Ele));
35     assert(pv != NULL);
36
37     pv->counter = 0;
38     pv->max = INIT_SIZE;
39     return pv;
40 }
```

6.12.3.2 void destory (PVector *pv*)

vector

PVector	

vector.c 48 .

```
49 {
50     assert(pv != NULL);
51     free(pv->array);
52     free(pv);
53
54     pv = NULL;
55 }
```

6.12.3.3 PEle get (PVector *pv,* size_t *index*)

PVectorindex

PVector	
index	

PEle, NULL

vector.c 290 .

```
291 {
292     if(pv == NULL || index > pv->counter)
293         return NULL;
294     return pv->array+index;
295 }
```

6.12.3.4 PEle get_back (PVector *pv*)

PVector	

PEle, NULL

vector.c 276 .

```
277 {
278     return get(pv, pv->counter-1);
279 }
```

6.12.3.5 PEle get_front (PVector *pv*)

PVector	

PEle, NULL

vector.c 262 .

```
263 {
264     return get(pv, 0);
265 }
```

6.12.3.6 int insert (PVector *pv*, size_t *index*, PEle *pe*)

———

—

PVector	
index	
PEle	

10

vector.c 138 .

```
139 {
140     if(pv == NULL || pe == NULL || index > pv->counter)
141         return 0;
142     enlarge(pv);
143
144 #if 0
145     for(size_t i = pv->counter; i > index; i--)
146         pv->array[i] = pv->array[i-1];
147     pv->array[i] = *pe;
148     pv->counter++;
149 #endif
150     memcpy(pv->array+index+1 ,pv->array+index, (pv->counter-index)*sizeof(Ele));
151     pv->array[index] = *pe;
152     pv->counter++;
153     return 1;
154 }
```

6.12.3.7 int mremove (PVector *pv,* size_t *index*)

index

PVector	
index	[0, counter -1]

10

vector.c 191 .

```
192 {
193     if(pv == NULL || index > pv->counter-1)
194         return 0;
195     //memcpy(pv->array+index, pv->array+index+1, pv->counter-index);    //error
196     int i = index;
197     for(; i < pv->counter-1; i++)
198         pv->array[i] = pv->array[i+1];
199     //*(pv->array+pv->counter) = 0;
200     //pv->counter--;
201     //*(pv->array+pv->counter--) = 0;   //error
202     memset(pv->array+pv->counter--, 0, sizeof(Ele));   //error
203     return 1;
204 }
```

6.12.3.8 int push_back (PVector *pv,* PEle *pe*)

PVector	

PEle	

10

vector.c 114 .

```
115 {
116     return insert(pv, pv->counter, pe);
117
118 #if 0
119      if(pv == NULL || pe == NULL)
120          return 0;
121      enlarge(pv);
122      pv->array[pv->counter++] = *pe;
123      //*(pv->array+pv->counter) = *pe;
124      //pv->counter++;
125      return 1;
126 #endif
127 }
```

6.12.3.9 int push_front (PVector *pv*, PEle *pe*)

PVector	
PEle	

1,0

vector.c 86 .

```
87 {
88     return insert(pv, 0, pe);
89
90 #if 0
91     if(pv == NULL || pe == NULL)
92         return 0;
93     enlarge(pv);
94
95     for(size_t i = pv->counter; i > 0; i--)
96         pv->array[i] = pv->array[i-1];
97     //memcpy(pv->array+1, pv->array, pv->counter);
98
99     pv->array[0] = *pe;
100     pv->counter++;
101     return 1;
102 #endif
103 }
```

6.12.3.10 int remove_back (PVector *pv*)

PVector	

10

vector.c 177 .

```
178 {
179     return mremove(pv, pv->counter-1);
180 }
```

6.12.3.11 int remove_front (PVector *pv*)

0

PVector	

10

vector.c 164 .

```
165 {
166     return mremove(pv, 0);
167 }
```

6.12.3.12 int set (PVector *pv*, size_t *index*, PEle *pe*)

PVectorindex*pe

PVector	
index	
PEle	

10

vector.c 243 .

```
244 {
245     if(pv == NULL || pe == NULL || index > pv->counter)
246         return 0;
247     pv->array[index] = *pe;
248     //memcpy(pv->array+index, pe, sizeof(*pe));
249     return 1;
250 }
```

6.12.3.13 int set_back (PVector *pv*, PEle *pe*)

PVector*pe

PVector	
PEle	

10

vector.c 228 .

```
229 {
230     return set(pv, pv->counter-1, pe);
231 }
```

6.12.3.14 int set_front (PVector *pv*, PEle *pe*)

PVector*pe

PVector	
PEle	

10

vector.c 214 .

```
215 {
216     return set(pv, 0, pe);
217 }
```

6.12.3.15 size_t size (PVector *pv*)

PVector counter

PVector	

-1

vector.c 305 .

```
306 {
307     if(pv == NULL)
308         return -1;
309
310     return pv->counter;
311 }
```