CircularDoublyLinkedList

Generated by Doxygen 1.8.11

Contents

1 Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

2 File Index

2.1 File List

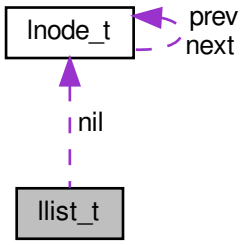
Here is a list of all files with brief descriptions:

3 Data Structure Documentation

3.1 llist_t Struct Reference

```
#include <llist.h>
```

Collaboration diagram for llist_t:



Data Fields

- size_t width
- lnode_t * nil
- int count

3.1.1 Detailed Description

Definition at line 22 of file llist.h.

3.1.2 Field Documentation

3.1.2.1 int count

count element amount

Definition at line 25 of file llist.h.

3.1.2.2 lnode_t* nil

sentinel (dummy node)

Definition at line 24 of file llist.h.

3.1.2.3 size_t width

element size (in bytes)

Definition at line 23 of file llist.h.

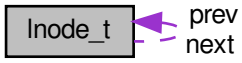
The documentation for this struct was generated from the following file:

- llist.h

3.2 lnode_t Struct Reference

`#include <llist.h>`

Collaboration diagram for lnode_t:



Data Fields

- struct lnode_t * prev
- void * data
- struct lnode_t * next

3.2.1 Detailed Description

Definition at line 16 of file llist.h.

3.2.2 Field Documentation

3.2.2.1 void* data

data pointer

Definition at line 18 of file llist.h.

3.2.2.2 struct lnode_t* next

next node

Definition at line 19 of file llist.h.

3.2.2.3 struct lnode_t* prev

previous node

Definition at line 17 of file llist.h.

The documentation for this struct was generated from the following file:

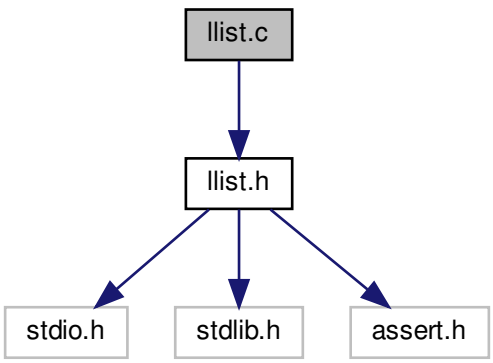
- llist.h

4 File Documentation

4.1 llist.c File Reference

```
#include "llist.h"
```

Include dependency graph for llist.c:



Functions

- llist_t * llist_create (size_t width)
- void llist_destruct (llist_t *l)
- lnode_t * llist_lsearch (llist_t *l, int n)
- void llist_delete (llist_t *l, int n)
- lnode_t * llist_insert (llist_t *l, int n, void *e)
- void llist_int_print (llist_t *l)

4.1.1 Function Documentation

4.1.1.1 llist_t* llist_create (size_t *width*)

Given the size of each element and the list size, create a list.

Parameters

width	size of each element
max_size	size of the list, max_size*width bytes will be reserved (definitively) for the list

Returns

a list initialized

Definition at line 10 of file llist.c.

```
00010                                   {
00011     llist_t* l = malloc(sizeof(llist_t));
00012     assert(l);
00013     l->nil = malloc(sizeof(lnode_t));
00014     assert(l->nil);
00015     l->nil->prev = l->nil;
00016     l->nil->next = l->nil;
00017     l->width = width;
00018     l->count = 0;
00019     return l;
00020 }
```

4.1.1.2 void llist_delete (llist_t * *l*, int *n*)

Definition at line 60 of file llist.c.

```
00060                                     {
00061     if (l->count == 0) return ;
00062     lnode_t* x =  llist_lsearch(l, n);
00063     x->prev->next = x->next;
00064     x->next->prev = x->prev;
00065     free(x->data);
00066     free(x);
00067     l->count--;
00068 }
```

Here is the call graph for this function:

4.1.1.3 void llist_destruct (llist_t * *l*)

Free a list.

Parameters

q	a list

Definition at line 27 of file llist.c.

```
00027                                   {
00028     lnode_t* x = l->nil->next;
00029     while(x != l->nil) {
00030         free(x->data);
00031         x = x->next;
00032         free(x->prev);
00033     }
00034     free(l->nil);
00035     free(l);
00036 }
```

4.1.1.4 lnode_t* llist_insert (llist_t * *l*, int *n*, void * *e*)

Definition at line 92 of file llist.c.

```
00092                                                           {
00093     lnode_t* x = l->count == 0 ? l->nil : llist_lsearch(l, n);
00094     lnode_t* node = malloc(sizeof(lnode_t));
00095     assert(node);
00096     node->data = e;
00097     llist_insert_ptr(x, node);
00098     l->count++;
00099     return node;
00100 }
```

Here is the call graph for this function:

4.1.1.5 void llist_int_print (llist_t * l)

Print an int list

Definition at line 106 of file llist.c.

```
00106                                   {
00107     printf("%d nodes : nil<->", l->count);
00108     lnode_t* x = l->nil->next;
00109     for(int i = 0; i < l->count; i++) {
00110         printf("[%d]<->", *((int*)x->data));
00111         x = x->next;
00112     }
00113     puts("nil");
00114 }
```

4.1.1.6 lnode_t* llist_lsearch (llist_t * l, int n)

Definition at line 45 of file llist.c.

```
00045                                              {
00046     assert (n >= 0 || n < l->count) ;
00047     lnode_t* x = l->nil->next;
00048     for(int i = 0; i < n; i++) {
00049         x = x->next;
00050     }
00051     return x;
00052 }
```

4.2 llist.c

```
00001 #include "llist.h"
00002
00010 llist_t* llist_create(size_t width) {
00011     llist_t* l = malloc(sizeof(llist_t));
00012     assert(l);
00013     l->nil = malloc(sizeof(lnode_t));
00014     assert(l->nil);
00015     l->nil->prev = l->nil;
00016     l->nil->next = l->nil;
00017     l->width = width;
00018     l->count = 0;
00019     return l;
00020 }
00021
00027 void llist_destruct(llist_t* l) {
00028     lnode_t* x = l->nil->next;
00029     while(x != l->nil) {
00030         free(x->data);
00031         x = x->next;
00032         free(x->prev);
00033     }
00034     free(l->nil);
00035     free(l);
00036 }
00037
00038 /*
00039  * Given an index n, do a linear search on a list
00040  * \param l a list
00041  * \param n index
00042  * \return the node of index n
00043  */
00044
00045 lnode_t* llist_lsearch(llist_t* l, int n) {
00046     assert (n >= 0 || n < l->count) ;
00047     lnode_t* x = l->nil->next;
00048     for(int i = 0; i < n; i++) {
00049         x = x->next;
00050     }
00051     return x;
00052 }
00053
00054 /*
```

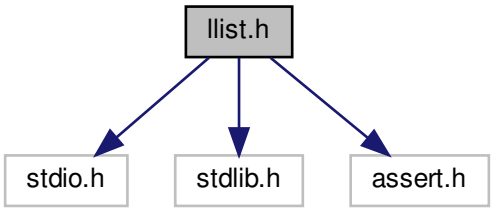
```
00055  * Delete the node of index n
00056  * \param l a list
00057  * \param n index
00058  */
00059
00060 void llist_delete(llist_t* l, int n) {
00061     if (l->count == 0) return ;
00062     lnode_t* x =  llist_lsearch(l, n);
00063     x->prev->next = x->next;
00064     x->next->prev = x->prev;
00065     free(x->data);
00066     free(x);
00067     l->count--;
00068 }
00069
00070 /*
00071  * Insert in the front of a given \e node a node \e x
00072  * \param node the node to prepend
00073  * \param x the node to insert
00074  */
00075
00076 static void llist_insert_ptr(lnode_t* node, lnode_t* x) {
00077     lnode_t* pn = node->prev;
00078     x->next = pn->next;
00079     pn->next->prev = x;
00080     pn->next = x;
00081     x->prev = pn;
00082 }
00083
00084 /*
00085  * Given an index n, insert in the front of the node n
00086  * \param l a list
00087  * \param n index
00088  * \param e element
00089  * \return the new node of index n which be inserted
00090  */
00091
00092 lnode_t* llist_insert(llist_t* l, int n, void* e) {
00093     lnode_t* x = l->count == 0 ? l->nil : llist_lsearch(l, n);
00094     lnode_t* node = malloc(sizeof(lnode_t));
00095     assert(node);
00096     node->data = e;
00097     llist_insert_ptr(x, node);
00098     l->count++;
00099     return node;
00100 }
00101
00106 void llist_int_print(llist_t* l) {
00107     printf("%d nodes : nil<->", l->count);
00108     lnode_t* x = l->nil->next;
00109     for(int i = 0; i < l->count; i++) {
00110         printf("[%d]<->", *((int*)x->data));
00111         x = x->next;
00112     }
00113     puts("nil");
00114 }
```

4.3 llist.h File Reference

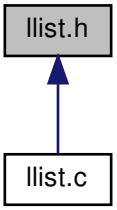
Circular doubly linked list definition and basic operations.

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

Include dependency graph for llist.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct lnode_t
- struct llist_t

Typedefs

- typedef struct lnode_t lnode_t
- typedef struct llist_t llist_t

Functions

- llist_t * llist_create (size_t width)
- void llist_destruct (llist_t *l)
- lnode_t * llist_insert (llist_t *l, int n, void *e)
- void llist_delete (llist_t *l, int n)
- void llist_int_print (llist_t *l)
- lnode_t * llist_lsearch (llist_t *l, int n)

4.3.1 Detailed Description

Circular doubly linked list definition and basic operations.

Author

Firmin MARTIN

Version

0.1

Date

01/01/2018

Definition in file llist.h.

4.3.2 Typedef Documentation

4.3.2.1 typedef struct llist_t llist_t

4.3.2.2 typedef struct lnode_t lnode_t

4.3.3 Function Documentation

4.3.3.1 llist_t* llist_create (size_t *width*)

Given the size of each element and the list size, create a list.

Parameters

width	size of each element
max_size	size of the list, max_size∗width bytes will be reserved (definitively) for the list

Returns

a list initialized

Definition at line 10 of file llist.c.

```
00010                                   {
00011     llist_t* l = malloc(sizeof(llist_t));
00012     assert(l);
00013     l->nil = malloc(sizeof(lnode_t));
00014     assert(l->nil);
00015     l->nil->prev = l->nil;
00016     l->nil->next = l->nil;
00017     l->width = width;
00018     l->count = 0;
00019     return l;
00020 }
```

4.3.3.2 void llist_delete (llist_t * l, int n)

Definition at line 60 of file llist.c.

```
00060                                       {
00061     if (l->count == 0) return ;
00062     lnode_t* x =  llist_lsearch(l, n);
00063     x->prev->next = x->next;
00064     x->next->prev = x->prev;
00065     free(x->data);
00066     free(x);
00067     l->count--;
00068 }
```

Here is the call graph for this function:

4.3.3.3 void llist_destruct (llist_t * l)

Free a list.

Parameters

q	a list

Definition at line 27 of file llist.c.

```
00027                                   {
00028     lnode_t* x = l->nil->next;
00029     while(x != l->nil) {
00030         free(x->data);
00031         x = x->next;
00032         free(x->prev);
00033     }
00034     free(l->nil);
00035     free(l);
00036 }
```

4.3.3.4 lnode_t* llist_insert (llist_t * l, int n, void * e)

Definition at line 92 of file llist.c.

```
00092                                                       {
00093     lnode_t* x = l->count == 0 ? l->nil : llist_lsearch(l, n);
00094     lnode_t* node = malloc(sizeof(lnode_t));
00095     assert(node);
00096     node->data = e;
00097     llist_insert_ptr(x, node);
00098     l->count++;
00099     return node;
00100 }
```

Here is the call graph for this function:

4.3.3.5 void llist_int_print (llist_t * *l*)

Print an int list

Definition at line 106 of file llist.c.

```
00106                                   {
00107     printf("%d nodes : nil<->", l->count);
00108     lnode_t* x = l->nil->next;
00109     for(int i = 0; i < l->count; i++) {
00110         printf("[%d]<->", *((int*)x->data));
00111         x = x->next;
00112     }
00113     puts("nil");
00114 }
```

4.3.3.6 lnode_t* llist_lsearch (llist_t * *l*, int *n*)

Definition at line 45 of file llist.c.

```
00045                                           {
00046     assert (n >= 0 || n < l->count) ;
00047     lnode_t* x = l->nil->next;
00048     for(int i = 0; i < n; i++) {
00049         x = x->next;
00050     }
00051     return x;
00052 }
```

4.4 llist.h

```
00001 #ifndef LLIST_H
00002 #define LLIST_H
00003
00004 #include <stdio.h>
00005 #include <stdlib.h>
00006 #include <assert.h>
00007
00016 typedef struct lnode_t {
00017     struct lnode_t* prev;
00018     void* data;
00019     struct lnode_t* next;
00020 } lnode_t;
00021
00022 typedef struct llist_t {
00023     size_t width;
00024     lnode_t* nil;
00025     int count;
00026 } llist_t;
00027
00028 llist_t*  llist_create(size_t width);
00029 void  llist_destruct(llist_t* l);
00030 lnode_t*  llist_insert(llist_t* l, int n, void* e);
00031 void llist_delete(llist_t* l, int n);
00032 void  llist_int_print(llist_t* l);
00033 lnode_t*  llist_lsearch(llist_t* l, int n);
00034
00035 #endif /* ifndef LLIST_H */
```

Index