LinkedList

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

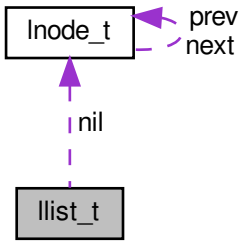
Here is a list of all files with brief descriptions:

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.

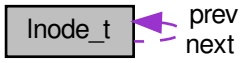
The documentation for this struct was generated from the following file:

- 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

Definition at line 18 of file llist.h.

3.2.2.2 struct lnode_t* next

Definition at line 19 of file llist.h.

3.2.2.3 struct lnode_t* prev

Definition at line 17 of file llist.h.

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

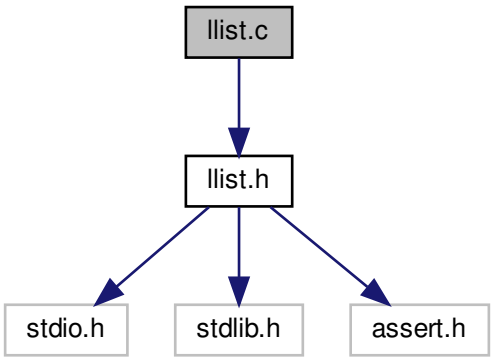
- llist.h

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*)

Definition at line 7 of file llist.c.

```
00007                                       {
00008     llist_t* l = malloc(sizeof(llist_t));
00009     assert(l);
00010     l->nil = malloc(sizeof(lnode_t));
00011     assert(l->nil);
00012     l->nil->prev = l->nil;
00013     l->nil->next = l->nil;
00014     l->width = width;
00015     l->count = 0;
00016     return l;
00017 }
```

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

Definition at line 39 of file llist.c.

```
00039                                        {
00040     if (l->count == 0) return ;
00041     lnode_t* x =  llist_lsearch(l, n);
00042     x->prev->next = x->next;
00043     x->next->prev = x->prev;
00044     free(x->data);
00045     free(x);
00046     l->count--;
00047 }
```

Here is the call graph for this function:

4.1.1.3 void llist_destruct (llist_t * *l*)

Definition at line 19 of file llist.c.

```
00019                                       {
00020     lnode_t* x = l->nil->next;
00021     while(x != l->nil) {
00022         free(x->data);
00023         x = x->next;
00024         free(x->prev);
00025     }
00026     free(l->nil);
00027     free(l);
00028 }
```

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

Definition at line 57 of file llist.c.

```
00057                                                   {
00058     lnode_t* x = l->count == 0 ? l->nil : llist_lsearch(l, n);
00059     lnode_t* node = malloc(sizeof(lnode_t));
00060     assert(node);
00061     node->data = e;
00062     llist_insert_ptr(x, node);
00063     l->count++;
00064     return node;
00065 }
```

Here is the call graph for this function:

4.1.1.5 void llist_int_print (llist_t * l)

Definition at line 67 of file llist.c.

```
00067                                   {
00068     printf("%d nodes : nil<->", l->count);
00069     lnode_t* x = l->nil->next;
00070     for(int i = 0; i < l->count; i++) {
00071         printf("[%d]<->", *((int*)x->data));
00072         x = x->next;
00073     }
00074     puts("nil");
00075 }
```

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

Definition at line 30 of file llist.c.

```
00030                                           {
00031     assert (n >= 0 || n < l->count) ;
00032     lnode_t* x = l->nil->next;
00033     for(int i = 0; i < n; i++) {
00034         x = x->next;
00035     }
00036     return x;
00037 }
```

4.2 llist.c

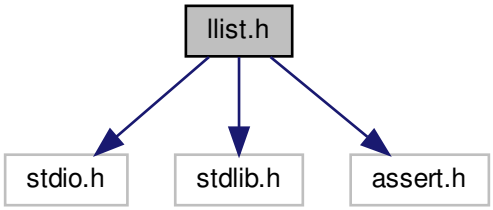
```
00001 #include "llist.h"
00002
00003 /*
00004  *
00005  */
00006
00007 llist_t* llist_create(size_t width) {
00008     llist_t* l = malloc(sizeof(llist_t));
00009     assert(l);
00010     l->nil = malloc(sizeof(lnode_t));
00011     assert(l->nil);
00012     l->nil->prev = l->nil;
00013     l->nil->next = l->nil;
00014     l->width = width;
00015     l->count = 0;
00016     return l;
00017 }
00018
00019 void llist_destruct(llist_t* l) {
00020     lnode_t* x = l->nil->next;
00021     while(x != l->nil) {
00022         free(x->data);
00023         x = x->next;
00024         free(x->prev);
00025     }
00026     free(l->nil);
00027     free(l);
00028 }
00029
00030 lnode_t* llist_lsearch(llist_t* l, int n) {
00031     assert (n >= 0 || n < l->count) ;
00032     lnode_t* x = l->nil->next;
00033     for(int i = 0; i < n; i++) {
00034         x = x->next;
00035     }
00036     return x;
00037 }
00038
00039 void llist_delete(llist_t* l, int n) {
00040     if (l->count == 0) return ;
00041     lnode_t* x =  llist_lsearch(l, n);
00042     x->prev->next = x->next;
00043     x->next->prev = x->prev;
00044     free(x->data);
00045     free(x);
00046     l->count--;
00047 }
00048
00049 static void llist_insert_ptr(lnode_t* node, lnode_t* x) {
00050     lnode_t* pn = node->prev;
00051     x->next = pn->next;
00052     pn->next->prev = x;
00053     pn->next = x;
00054     x->prev = pn;
00055 }
00056
00057 lnode_t* llist_insert(llist_t* l, int n, void* e) {
00058     lnode_t* x = l->count == 0 ? l->nil : llist_lsearch(l, n);
00059     lnode_t* node = malloc(sizeof(lnode_t));
00060     assert(node);
00061     node->data = e;
00062     llist_insert_ptr(x, node);
00063     l->count++;
00064     return node;
00065 }
00066
00067 void llist_int_print(llist_t* l) {
00068     printf("%d nodes : nil<->", l->count);
00069     lnode_t* x = l->nil->next;
00070     for(int i = 0; i < l->count; i++) {
00071         printf("[%d]<->", *((int*)x->data));
00072         x = x->next;
00073     }
00074     puts("nil");
00075 }
```

4.3 llist.h File Reference

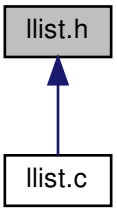
Circular doubly linked list definition and basic operations.

```
#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*)

Definition at line 7 of file llist.c.

```
00007                                    {
00008     llist_t* l = malloc(sizeof(llist_t));
00009     assert(l);
00010     l->nil = malloc(sizeof(lnode_t));
00011     assert(l->nil);
00012     l->nil->prev = l->nil;
00013     l->nil->next = l->nil;
00014     l->width = width;
00015     l->count = 0;
00016     return l;
00017 }
```

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

Definition at line 39 of file llist.c.

```
00039                                      {
00040     if (l->count == 0) return ;
00041     lnode_t* x =  llist_lsearch(l, n);
00042     x->prev->next = x->next;
00043     x->next->prev = x->prev;
00044     free(x->data);
00045     free(x);
00046     l->count--;
00047 }
```

Here is the call graph for this function:

4.3.3.3 void llist_destruct (llist_t * l)

Definition at line 19 of file llist.c.

```
00019                                   {
00020     lnode_t* x = l->nil->next;
00021     while(x != l->nil) {
00022         free(x->data);
00023         x = x->next;
00024         free(x->prev);
00025     }
00026     free(l->nil);
00027     free(l);
00028 }
```

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

Definition at line 57 of file llist.c.

```
00057                                                       {
00058     lnode_t* x = l->count == 0 ? l->nil : llist_lsearch(l, n);
00059     lnode_t* node = malloc(sizeof(lnode_t));
00060     assert(node);
00061     node->data = e;
00062     llist_insert_ptr(x, node);
00063     l->count++;
00064     return node;
00065 }
```

Here is the call graph for this function:

4.3.3.5 void llist_int_print (llist_t * l)

Definition at line 67 of file llist.c.

```
00067                                    {
00068     printf("%d nodes : nil<->", l->count);
00069     lnode_t* x = l->nil->next;
00070     for(int i = 0; i < l->count; i++) {
00071         printf("[%d]<->", *((int*)x->data));
00072         x = x->next;
00073     }
00074     puts("nil");
00075 }
```

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

Definition at line 30 of file llist.c.

```
00030                                           {
00031     assert (n >= 0 || n < l->count) ;
00032     lnode_t* x = l->nil->next;
00033     for(int i = 0; i < n; i++) {
00034         x = x->next;
00035     }
00036     return x;
00037 }
```

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