# LinkedList

Generated by Doxygen 1.8.11

# Contents

# 1 Data Structure Index

## 1.1 Data Structures

Here are the data structures with brief descriptions:

| | |
|---|---|
| **llist_t** | **2** |

# 2 File Index

## 2.1 File List

Here is a list of all files with brief descriptions:

**llist.c** 4

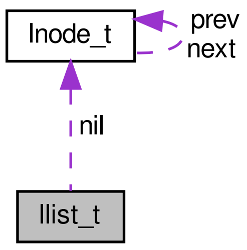
**llist.h**
**Doubly circular linked list definition and basic operations** 7

# 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.

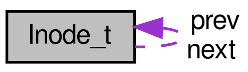
The documentation for this struct was generated from the following file:

- 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:

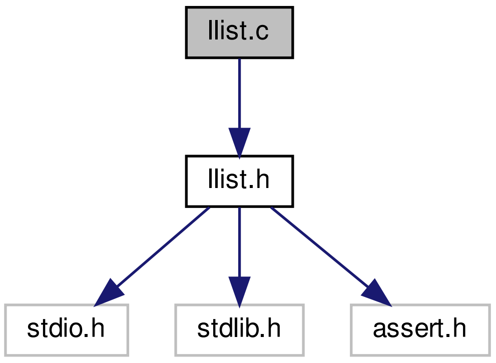
- llist.h

# 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 3 of file llist.c.

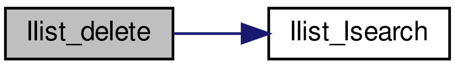
```
00003                                       {
00004     llist_t* l = malloc(sizeof(llist_t));
00005     assert(l);
00006     l->nil = malloc(sizeof(lnode_t));
00007     assert(l->nil);
00008     l->nil->prev = l->nil;
00009     l->nil->next = l->nil;
00010     l->width = width;
00011     l->count = 0;
00012     return l;
00013 }
```

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

Definition at line 35 of file llist.c.

```
00035                                        {
00036     if (l->count == 0) return ;
00037     lnode_t* x =  llist_lsearch(l, n);
00038     x->prev->next = x->next;
00039     x->next->prev = x->prev;
00040     free(x->data);
00041     free(x);
00042     l->count--;
00043 }
```

Here is the call graph for this function:



#### 4.1.1.3 void llist_destruct ( llist_t ∗ *l* )

Definition at line 15 of file llist.c.

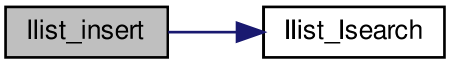
```
00015                                   {
00016     lnode_t* x = l->nil->next;
00017     while(x != l->nil) {
00018         free(x->data);
00019         x = x->next;
00020         free(x->prev);
00021     }
00022     free(l->nil);
00023     free(l);
00024 }
```

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

Definition at line 53 of file llist.c.

```
                                                          {
    lnode_t* x = l->count == 0 ? l->nil : llist_lsearch(l, n);
    lnode_t* node = malloc(sizeof(lnode_t));
    assert(node);
    node->data = e;
    llist_insert_ptr(x, node);
    l->count++;
    return node;
}
```

Here is the call graph for this function:



#### 4.1.1.5 void llist_int_print ( llist_t ∗ *l* )

Definition at line 63 of file llist.c.

```
                                     {
    printf("%d nodes : nil<->", l->count);
    lnode_t* x = l->nil->next;
    for(int i = 0; i < l->count; i++) {
        printf("[%d]<->", *((int*)x->data));
        x = x->next;
    }
    puts("nil");
}
```

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

Definition at line 26 of file llist.c.

```
                                                {
    assert (n >= 0 || n < l->count) ;
    lnode_t* x = l->nil->next;
    for(int i = 0; i < n; i++) {
        x = x->next;
    }
    return x;
}
```

## 4.2 llist.c

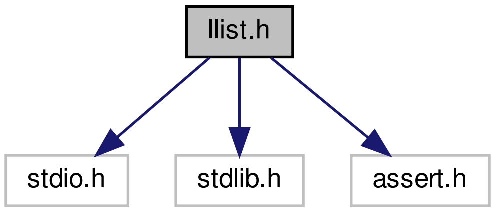
```c
#include "llist.h"

llist_t* llist_create(size_t width) {
    llist_t* l = malloc(sizeof(llist_t));
    assert(l);
    l->nil = malloc(sizeof(lnode_t));
    assert(l->nil);
    l->nil->prev = l->nil;
    l->nil->next = l->nil;
    l->width = width;
    l->count = 0;
    return l;
}

void llist_destruct(llist_t* l) {
    lnode_t* x = l->nil->next;
    while(x != l->nil) {
        free(x->data);
        x = x->next;
        free(x->prev);
    }
    free(l->nil);
    free(l);
}

lnode_t* llist_lsearch(llist_t* l, int n) {
    assert (n >= 0 || n < l->count) ;
    lnode_t* x = l->nil->next;
    for(int i = 0; i < n; i++) {
        x = x->next;
    }
    return x;
}

void llist_delete(llist_t* l, int n) {
    if (l->count == 0) return ;
    lnode_t* x =  llist_lsearch(l, n);
    x->prev->next = x->next;
    x->next->prev = x->prev;
    free(x->data);
    free(x);
    l->count--;
}

static void llist_insert_ptr(lnode_t* node, lnode_t* x) {
    lnode_t* pn = node->prev;
    x->next = pn->next;
    pn->next->prev = x;
    pn->next = x;
    x->prev = pn;
}

lnode_t* llist_insert(llist_t* l, int n, void* e) {
    lnode_t* x = l->count == 0 ? l->nil : llist_lsearch(l, n);
    lnode_t* node = malloc(sizeof(lnode_t));
    assert(node);
    node->data = e;
    llist_insert_ptr(x, node);
    l->count++;
    return node;
}

void llist_int_print(llist_t* l) {
    printf("%d nodes : nil<->", l->count);
    lnode_t* x = l->nil->next;
    for(int i = 0; i < l->count; i++) {
        printf("[%d]<->", *((int*)x->data));
        x = x->next;
    }
    puts("nil");
}
```

## 4.3 llist.h File Reference

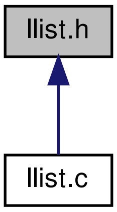
Doubly circular linked list definition and basic operations.

```
#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

Doubly circular 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 3 of file llist.c.

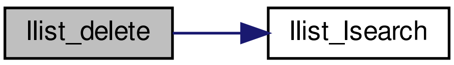
```
00003                                      {
00004     llist_t* l = malloc(sizeof(llist_t));
00005     assert(l);
00006     l->nil = malloc(sizeof(lnode_t));
00007     assert(l->nil);
00008     l->nil->prev = l->nil;
00009     l->nil->next = l->nil;
00010     l->width = width;
00011     l->count = 0;
00012     return l;
00013 }
```

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

Definition at line 35 of file llist.c.

```
00035                                       {
00036     if (l->count == 0) return ;
00037     lnode_t* x =  llist_lsearch(l, n);
00038     x->prev->next = x->next;
00039     x->next->prev = x->prev;
00040     free(x->data);
00041     free(x);
00042     l->count--;
00043 }
```

Here is the call graph for this function:



#### 4.3.3.3 void llist_destruct ( llist_t ∗ *l* )

Definition at line 15 of file llist.c.

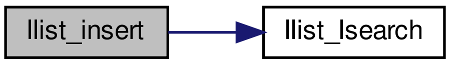
```
                               {
    lnode_t* x = l->nil->next;
    while(x != l->nil) {
        free(x->data);
        x = x->next;
        free(x->prev);
    }
    free(l->nil);
    free(l);
}
```

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

Definition at line 53 of file llist.c.

```
                                                 {
    lnode_t* x = l->count == 0 ? l->nil : llist_lsearch(l, n);
    lnode_t* node = malloc(sizeof(lnode_t));
    assert(node);
    node->data = e;
    llist_insert_ptr(x, node);
    l->count++;
    return node;
}
```

Here is the call graph for this function:



#### 4.3.3.5 void llist_int_print ( llist_t ∗ *l* )

Definition at line 63 of file llist.c.

```
                                {
    printf("%d nodes : nil<->", l->count);
    lnode_t* x = l->nil->next;
    for(int i = 0; i < l->count; i++) {
        printf("[%d]<->", *((int*)x->data));
        x = x->next;
    }
    puts("nil");
}
```

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

Definition at line 26 of file llist.c.

```
00026                                          {
00027     assert (n >= 0 || n < l->count) ;
00028     lnode_t* x = l->nil->next;
00029     for(int i = 0; i < n; i++) {
00030         x = x->next;
00031     }
00032     return x;
00033 }
```

## 4.4 llist.h

```
#ifndef LLIST_H
#define LLIST_H

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

typedef struct lnode_t {
    struct lnode_t* prev;
    void* data;
    struct lnode_t* next;
} lnode_t;

typedef struct llist_t {
    size_t width;
    lnode_t* nil;
    int count;
} llist_t;

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);

#endif /* ifndef LLIST_H */
```

# Index