

tAltris
v1.0

Generated by Doxygen 1.8.13

Contents

| | | |
|----------|--------------------------------------|----------|
| 1 | Data Structure Index | 1 |
| 1.1 | Data Structures | 1 |
| 2 | File Index | 3 |
| 2.1 | File List | 3 |
| 3 | Data Structure Documentation | 5 |
| 3.1 | list Struct Reference | 5 |
| 3.1.1 | Detailed Description | 5 |
| 3.1.2 | Field Documentation | 5 |
| 3.1.2.1 | first | 6 |
| 3.1.2.2 | length | 6 |
| 3.2 | list_node Struct Reference | 6 |
| 3.2.1 | Detailed Description | 6 |
| 3.2.2 | Field Documentation | 6 |
| 3.2.2.1 | next | 6 |

| | |
|---|----------|
| 4 File Documentation | 7 |
| 4.1 src/tAltris.c File Reference | 7 |
| 4.1.1 Function Documentation | 7 |
| 4.1.1.1 main() | 7 |
| 4.2 src/tAltris.h File Reference | 8 |
| 4.3 src/utlis/list.c File Reference | 8 |
| 4.3.1 Function Documentation | 9 |
| 4.3.1.1 list_add() | 9 |
| 4.3.1.2 list_advance() | 9 |
| 4.3.1.3 list_append() | 10 |
| 4.3.1.4 list_at() | 11 |
| 4.3.1.5 list_concat() | 12 |
| 4.3.1.6 list_del() | 13 |
| 4.3.1.7 list_del_after() | 13 |
| 4.3.1.8 list_del_at() | 14 |
| 4.3.1.9 list_first() | 14 |
| 4.3.1.10 list_init() | 15 |
| 4.3.1.11 list_insert_after() | 15 |
| 4.3.1.12 list_insert_at() | 16 |
| 4.3.1.13 list_is_empty() | 16 |
| 4.3.1.14 list_last() | 17 |
| 4.3.1.15 list_length() | 17 |
| 4.3.1.16 list_next() | 18 |
| 4.3.1.17 list_print() | 18 |
| 4.3.1.18 list_reverse() | 19 |
| 4.3.1.19 list_sort() | 19 |
| 4.3.1.20 list_split_at() | 20 |
| 4.3.1.21 list_swap() | 21 |
| 4.4 src/utlis/list.h File Reference | 21 |
| 4.4.1 Detailed Description | 23 |

| | | |
|----------|--|----|
| 4.4.2 | Macro Definition Documentation | 23 |
| 4.4.2.1 | list_elt | 23 |
| 4.4.2.2 | list_foreach | 23 |
| 4.4.2.3 | list_foreach_elt | 24 |
| 4.4.2.4 | list_foreach_elt_safe | 25 |
| 4.4.2.5 | list_foreach_safe | 25 |
| 4.4.3 | Function Documentation | 26 |
| 4.4.3.1 | list_add() | 26 |
| 4.4.3.2 | list_advance() | 27 |
| 4.4.3.3 | list_append() | 27 |
| 4.4.3.4 | list_at() | 28 |
| 4.4.3.5 | list_concat() | 28 |
| 4.4.3.6 | list_del() | 29 |
| 4.4.3.7 | list_del_after() | 29 |
| 4.4.3.8 | list_del_at() | 30 |
| 4.4.3.9 | list_first() | 31 |
| 4.4.3.10 | list_init() | 31 |
| 4.4.3.11 | list_insert_after() | 32 |
| 4.4.3.12 | list_insert_at() | 32 |
| 4.4.3.13 | list_is_empty() | 33 |
| 4.4.3.14 | list_last() | 33 |
| 4.4.3.15 | list_length() | 34 |
| 4.4.3.16 | list_next() | 34 |
| 4.4.3.17 | list_print() | 35 |
| 4.4.3.18 | list_reverse() | 35 |
| 4.4.3.19 | list_sort() | 36 |
| 4.4.3.20 | list_split_at() | 36 |
| 4.4.3.21 | list_swap() | 37 |

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

| | |
|------------------|---|
| list | 5 |
| list_node | 6 |

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

| | |
|---|----|
| src/ tAltris.c | 7 |
| src/ tAltris.h | 8 |
| src/utls/ list.c | 8 |
| src/utls/ list.h Intrusive list implement | 21 |

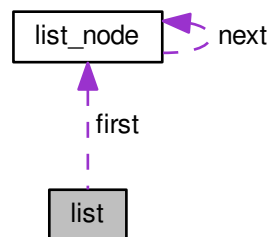
Chapter 3

Data Structure Documentation

3.1 list Struct Reference

```
#include <list.h>
```

Collaboration diagram for list:



Data Fields

- `size_t` **length**
- `struct list_node *` **first**

3.1.1 Detailed Description

Head of a singly-linked list.

3.1.2 Field Documentation

3.1.2.1 first

```
struct list_node* first
```

3.1.2.2 length

```
size_t length
```

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

- src/utils/ **list.h**

3.2 list_node Struct Reference

```
#include <list.h>
```

Collaboration diagram for list_node:



Data Fields

- struct **list_node** * **next**

3.2.1 Detailed Description

A node of a singly-linked list.

3.2.2 Field Documentation

3.2.2.1 next

```
struct list_node* next
```

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

- src/utils/ **list.h**

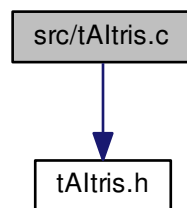
Chapter 4

File Documentation

4.1 src/tAltris.c File Reference

```
#include "tAltris.h"
```

Include dependency graph for tAltris.c:



Functions

- int **main** ()

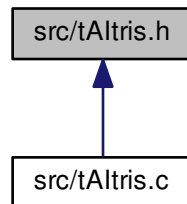
4.1.1 Function Documentation

4.1.1.1 main()

```
int main ( )
```

4.2 src/tAltris.h File Reference

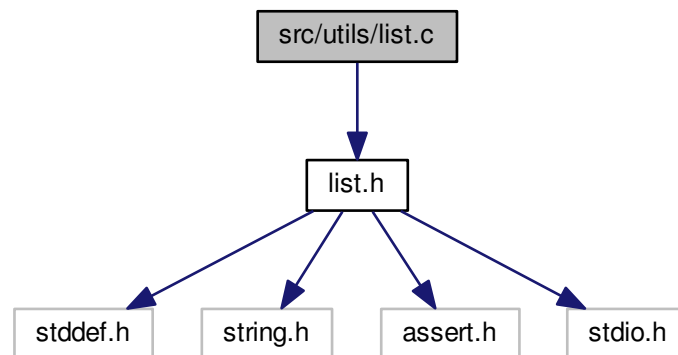
This graph shows which files directly or indirectly include this file:



4.3 src/utlis/list.c File Reference

```
#include "list.h"
```

Include dependency graph for list.c:



Functions

- void **list_init** (struct **list** * **list**)
- size_t **list_length** (const struct **list** * **list**)
- struct **list_node** * **list_first** (const struct **list** * **list**)
- struct **list_node** * **list_last** (const struct **list** * **list**)
- struct **list_node** * **list_next** (const struct **list_node** *node)
- struct **list_node** * **list_advance** (struct **list_node** *node, size_t distance)
- struct **list_node** * **list_at** (const struct **list** * **list**, size_t pos)

- void **list_reverse** (struct **list** * **list**)
- void **list_swap** (struct **list** *l1, struct **list** *l2)
- void **list_split_at** (struct **list** * **list**, size_t pos, struct **list** *right)
- void **list_concat** (struct **list** *l1, struct **list** *l2)
- void **list_sort** (struct **list** * **list**, int(*cmp)(struct **list_node** *, struct **list_node** *))
- int **list_is_empty** (const struct **list** * **list**)
- void **list_add** (struct **list** * **list**, struct **list_node** *node)
- void **list_append** (struct **list** * **list**, struct **list_node** *node)
- void **list_insert_after** (struct **list** * **list**, struct **list_node** *curr, struct **list_node** *node)
- void **list_insert_at** (struct **list** * **list**, struct **list_node** *node, size_t pos)
- void **list_del** (struct **list** * **list**)
- void **list_del_after** (struct **list** * **list**, struct **list_node** *node)
- void **list_del_at** (struct **list** * **list**, size_t pos)
- void **list_print** (struct **list** * **list**)

4.3.1 Function Documentation

4.3.1.1 list_add()

```
void list_add (
    struct list * list,
    struct list_node * node ) [inline]
```

Adds *node* in the front of *list*

Parameters

| | |
|-------------|---------------|
| <i>list</i> | a list. |
| <i>node</i> | the new node. |

Precondition

list must be not NULL.
node must be not NULL.

Postcondition

List size increases by 1.

Remarks

Complexity: O(1)

4.3.1.2 list_advance()

```
struct list_node* list_advance (
    struct list_node * node,
    size_t distance )
```

Returns the *nth*-node after the current one.

Parameters

| | |
|-----------------|----------------------|
| <i>node</i> | a node. |
| <i>distance</i> | distance to move on. |

Returns

the nth-node after *node*.

Precondition

node must be not NULL.

Remarks

Complexity: O(n)

4.3.1.3 list_append()

```
void list_append (
    struct list * list,
    struct list_node * node ) [inline]
```

Adds *node* at the end of *list*.

Parameters

| | |
|-------------|---------------|
| <i>list</i> | a list. |
| <i>node</i> | the new node. |

Precondition

list must be not NULL.
node must be not NULL.

Postcondition

List size increases by 1.

Remarks

Complexity: O(n)

4.3.1.4 list_at()

```
struct list_node* list_at (  
    const struct list * list,  
    size_t pos )
```

Returns node at the position *pos*.

Parameters

| | |
|-------------|---------------------------------|
| <i>list</i> | a list. |
| <i>pos</i> | position (0-based) of the node. |

Returns

the node at the position `pos`.

Precondition

`list` must be not NULL.
`list` must be not empty.
`pos` must be in `[0; list_length(list)[`.

Remarks

Complexity: O(N)

4.3.1.5 list_concat()

```
void list_concat (
    struct list * l1,
    struct list * l2 ) [inline]
```

Concatenates two lists.

Parameters

| | |
|-----------|---------|
| <i>l1</i> | list 1. |
| <i>l2</i> | list 2. |

Precondition

`l1` must be not NULL.
`l2` must be not NULL.
`l1` must be different of `l2`.

Postcondition

`l2` is reset to an empty list.

Remarks

Complexity: O(N)

4.3.1.6 list_del()

```
void list_del (
    struct list * list ) [inline]
```

Deletes the first node.

Parameters

| | |
|-------------|---------|
| <i>list</i> | a list. |
|-------------|---------|

Precondition

list must be not NULL.
list must be not empty.

Postcondition

List size decreases by 1.

Remarks

Complexity: O(1)

4.3.1.7 list_del_after()

```
void list_del_after (
    struct list * list,
    struct list_node * node ) [inline]
```

Deletes the node at after the node *curr*.

Parameters

| | |
|-------------|-------------------------|
| <i>list</i> | a list. |
| <i>node</i> | a node of <i>list</i> . |

Precondition

list must be not NULL.
node must be not NULL.
list must be not empty.
node must a node of *list*.

Postcondition

List size decreases by 1.

Remarks

Complexity: O(1)

4.3.1.8 list_del_at()

```
void list_del_at (
    struct list * list,
    size_t pos ) [inline]
```

Deletes the node at the position `pos`.

Parameters

| | |
|-------------|--|
| <i>list</i> | a list. |
| <i>pos</i> | index (0-based) of the node to delete. |

Precondition

`list` must be not NULL.
`list` must be not empty.
`pos` must be in `[0; list_length(list)[`.

Postcondition

List size decreases by 1.

Remarks

Complexity: O(n)

4.3.1.9 list_first()

```
struct list_node* list_first (
    const struct list * list )
```

Returns the first node.

Parameters

| | |
|-------------|---------|
| <i>list</i> | a list. |
|-------------|---------|

Returns

the first node.

Precondition

`list` must be not NULL.
`list` must be not empty.

Remarks

Complexity: O(1)

4.3.1.10 list_init()

```
void list_init (  
    struct list * list ) [inline]
```

Initializes the list.

Parameters

| | |
|-------------|---------|
| <i>list</i> | a list. |
|-------------|---------|

Precondition

`list` must be not NULL.

Postcondition

`list` is empty.
`list` has a size of 0.

Remarks

Complexity: O(1)

4.3.1.11 list_insert_after()

```
void list_insert_after (  
    struct list * list,  
    struct list_node * curr,  
    struct list_node * node ) [inline]
```

Inserts `node` at after the node `curr`.

Parameters

| | |
|-------------|-------------------------------|
| <i>list</i> | a list. |
| <i>curr</i> | a node of <code>list</code> . |
| <i>node</i> | new node. |

Precondition

`list` must be not NULL.
`curr` must be not NULL.
`curr` must a node of `list`.
`node` must be not NULL.

Postcondition

List size increases by 1.

Remarks

Complexity: O(1)

4.3.1.12 list_insert_at()

```
void list_insert_at (
    struct list * list,
    struct list_node * node,
    size_t pos ) [inline]
```

Inserts `node` at the position `pos` in `list`.

Parameters

| | |
|-------------|--|
| <i>list</i> | a list. |
| <i>node</i> | new node. |
| <i>pos</i> | position (0-based) where to insert the new node. |

Precondition

`list` must be not NULL.
`node` must be not NULL.
`pos` must be in `[0; list_length(list)]`.

Postcondition

List size increases by 1.

Remarks

Complexity: O(n)

4.3.1.13 list_is_empty()

```
int list_is_empty (
    const struct list * list ) [inline]
```

Tests if a list is empty.

Parameters

| | |
|-------------|---------|
| <i>list</i> | a list. |
|-------------|---------|

Returns

1 if the list is empty, otherwise 0.

Precondition

`list` must be not NULL.

Remarks

Complexity: O(1)

4.3.1.14 list_last()

```
struct list_node* list_last (  
    const struct list * list )
```

Returns the last node.

Parameters

| | |
|-------------|---------|
| <i>list</i> | a list. |
|-------------|---------|

Returns

the last node.

Precondition

`list` must be not NULL.

Remarks

Complexity: O(N)

4.3.1.15 list_length()

```
size_t list_length (  
    const struct list * list ) [inline]
```

Returns the size of the list.

Parameters

| | |
|-------------|---------|
| <i>list</i> | a list. |
|-------------|---------|

Returns

the length of the list.

Precondition

`list` must be not NULL.

Remarks

Complexity: O(1)

4.3.1.16 list_next()

```
struct list_node* list_next (
    const struct list_node * node )
```

Returns the next node.

Parameters

| | |
|-------------|---------|
| <i>node</i> | a node. |
|-------------|---------|

Returns

the next node.

Precondition

`node` must be not NULL.

Remarks

Complexity: O(1)

4.3.1.17 list_print()

```
void list_print (
    struct list * list )
```

Print the list

Parameters

| | |
|-------------|--------|
| <i>list</i> | a list |
|-------------|--------|

4.3.1.18 list_reverse()

```
void list_reverse (
    struct list * list ) [inline]
```

Reverses the order of the elements in the list.

Parameters

| | |
|-------------|---------|
| <i>list</i> | a list. |
|-------------|---------|

Precondition

list must be not NULL.

Remarks

Complexity: O(N)

4.3.1.19 list_sort()

```
void list_sort (
    struct list * list,
    int (*) (struct list_node *, struct list_node *) cmp ) [inline]
```

Sort a list using a comparison function.

The contents of the list are sorted in ascending order according to a comparison function which is called with two arguments that point to the node being compared.

The comparison function must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second.

If two members compare as equal, their order in the sorted list is preserved.

Parameters

| | |
|-------------|-----------------------------|
| <i>list</i> | list to sort. |
| <i>cmp</i> | comparison function to use. |

Precondition

`list` must be not NULL.
`cmp` must be not NULL.

Remarks

The sort is stable.
 Complexity: $O(N \log N)$
 Space complexity: $O(1)$

4.3.1.20 list_split_at()

```
void list_split_at (
    struct list * list,
    size_t pos,
    struct list * right ) [inline]
```

Splits a list in two parts at the position `pos`.

After the split:

- `list` contains nodes in `[0, pos[`
- `right` contains nodes in `[pos,length(list)[`

Examples:

```
list = [1, 2, 3]
list_split_at(list, 0, right) => ([], [1,2,3])
list_split_at(list, 1, right) => ([1], [2,3])
list_split_at(list, 2, right) => ([1,2], [3])
list_split_at(list, 3, right) => ([1,2,3], [])
list = []
list_split_at(list, 0, right) => ([], [])
```

Parameters

| | |
|--------------|--|
| <i>list</i> | list to split. |
| <i>pos</i> | position (0-based) where to split the list. |
| <i>right</i> | an empty list to receive the part after <code>pos</code> |

Precondition

`list` must be not NULL.
`right` must be not NULL.
`right` must be empty.
`list` must be different of `right`.

Remarks

Complexity: $O(N)$

4.3.1.21 list_swap()

```
void list_swap (
    struct list * l1,
    struct list * l2 ) [inline]
```

Swaps two lists.

Parameters

| | |
|-----------|---------|
| <i>l1</i> | list 1. |
| <i>l2</i> | list 2. |

Precondition

l1 must be not NULL.
l2 must be not NULL.
l1 must be different of *l2*.

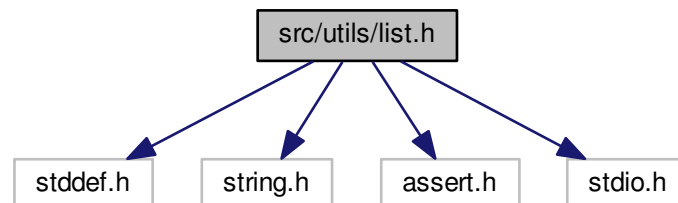
Remarks

Complexity: $O(1)$

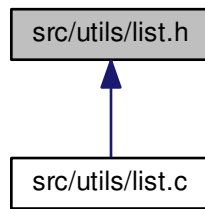
4.4 src/utls/list.h File Reference

Intrusive list implement.

```
#include <stddef.h>
#include <string.h>
#include <assert.h>
#include <stdio.h>
Include dependency graph for list.h:
```



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



Data Structures

- struct **list**
- struct **list_node**

Macros

- `#define list_elt(node, type, fieldname) ((type*)((char*)(node) - offsetof(type, fieldname)))`
- `#define list_foreach(list, curr) for (curr = list_first(list); curr != NULL; curr = list_next(curr))`
- `#define list_foreach_elt(list, curr, type, fieldname)`
- `#define list_foreach_safe(list, curr, tmp)`
- `#define list_foreach_elt_safe(list, curr, tmp, type, fieldname)`

Functions

- void **list_init** (struct **list** * **list**)
- size_t **list_length** (const struct **list** * **list**)
- struct **list_node** * **list_first** (const struct **list** * **list**)
- struct **list_node** * **list_last** (const struct **list** * **list**)
- struct **list_node** * **list_next** (const struct **list_node** *node)
- struct **list_node** * **list_advance** (struct **list_node** *node, size_t distance)
- struct **list_node** * **list_at** (const struct **list** * **list**, size_t pos)
- void **list_reverse** (struct **list** * **list**)
- void **list_swap** (struct **list** *l1, struct **list** *l2)
- void **list_split_at** (struct **list** * **list**, size_t pos, struct **list** *right)
- void **list_concat** (struct **list** *l1, struct **list** *l2)
- void **list_sort** (struct **list** * **list**, int(*cmp)(struct **list_node** *, struct **list_node** *))
- int **list_is_empty** (const struct **list** * **list**)
- void **list_add** (struct **list** * **list**, struct **list_node** *node)
- void **list_append** (struct **list** * **list**, struct **list_node** *node)
- void **list_insert_after** (struct **list** * **list**, struct **list_node** *curr, struct **list_node** *node)
- void **list_insert_at** (struct **list** * **list**, struct **list_node** *node, size_t pos)
- void **list_del** (struct **list** * **list**)
- void **list_del_after** (struct **list** * **list**, struct **list_node** *node)
- void **list_del_at** (struct **list** * **list**, size_t pos)
- void **list_print** (struct **list** * **list**)

4.4.1 Detailed Description

Intrusive list implement.

Author

S4MasterRace

Version

1.0

4.4.2 Macro Definition Documentation

4.4.2.1 list_elt

```
#define list_elt(  
    node,  
    type,  
    fieldname ) ((type*)((char*)(node) - offsetof(type, fieldname)))
```

Returns a pointer to the structure which contains the node.

Parameters

| | |
|------------------|---|
| <i>node</i> | a list node (struct list_node*). |
| <i>type</i> | type of the structure which contains the node. |
| <i>fieldname</i> | name of the node (field name) in the structure. |

Precondition

node must be not NULL.

Remarks

Complexity: O(1)

4.4.2.2 list_foreach

```
#define list_foreach(  
    list,  
    curr ) for (curr = list_first( list); curr != NULL; curr = list_next(curr))
```

Iterates over list (nodes).

Parameters

| | |
|-------------|---|
| <i>list</i> | a list (struct list*). |
| <i>curr</i> | a struct list_node* used to hold the current element. |

Precondition

`list` must be not NULL.
`curr` must be not NULL.

Remarks

Complexity: O(N)

4.4.2.3 list_foreach_elt

```
#define list_foreach_elt(
    list,
    curr,
    type,
    fieldname )
```

Value:

```
for (curr = list_elt(list_first(list), type, fieldname); \
     curr != NULL; \
     curr = curr->fieldname.next == NULL ? NULL : \
     list_elt(list_next(&(curr->fieldname)), type, fieldname))
```

Iterates over list (elements)

Parameters

| | |
|------------------|---|
| <i>list</i> | a list (struct list*). |
| <i>curr</i> | pointer (type*) used to hold the current element. |
| <i>type</i> | type of the structure which contains the node. |
| <i>fieldname</i> | name of the node (field name) in the structure. |

Precondition

`list` must be not NULL.
`list` must be not empty.
`curr` must be not NULL.

Remarks

Complexity: O(N)

4.4.2.4 list_foreach_elt_safe

```
#define list_foreach_elt_safe(
    list,
    curr,
    tmp,
    type,
    fieldname )
```

Value:

```
for (curr = list_elt(list_first(list), type, fieldname), \
    tmp = list_next(&(curr->fieldname)); \
    curr != NULL; \
    curr = tmp == NULL ? NULL : list_elt(tmp, type, fieldname), \
    tmp = tmp == NULL ? NULL : list_next(tmp)) \
```

Iterates over list (elements), allows deletion of the current element.

Parameters

| | |
|------------------|---|
| <i>list</i> | a list (struct list*). |
| <i>curr</i> | pointer (type*) used to hold the current element. |
| <i>tmp</i> | a struct list_node* used as temporary storage. |
| <i>type</i> | type of the structure which contains the node. |
| <i>fieldname</i> | name of the node (field name) in the structure. |

Precondition

```
list must be not NULL.
list must be not empty.
curr must be not NULL.
```

Remarks

Complexity: O(N)

4.4.2.5 list_foreach_safe

```
#define list_foreach_safe(
    list,
    curr,
    tmp )
```

Value:

```
for (curr = list_first(list), tmp = list_next(curr); \
    curr != NULL; \
    curr = tmp, tmp = tmp == NULL ? NULL : list_next(tmp)) \
```

Iterates over list (nodes), allows deletion of the current node.

Parameters

| | |
|-------------|---|
| <i>list</i> | a list (struct list*). |
| <i>curr</i> | a struct list_node* used to hold the current element. |
| <i>tmp</i> | a struct list_node* used as temporary storage. |

Precondition

`list` must be not NULL.
`curr` must be not NULL.
`tmp` must be not NULL.

Remarks

Complexity: O(N)

4.4.3 Function Documentation**4.4.3.1 list_add()**

```
void list_add (  
    struct list * list,  
    struct list_node * node ) [inline]
```

Adds `node` in the front of `list`

Parameters

| | |
|-------------|---------------|
| <i>list</i> | a list. |
| <i>node</i> | the new node. |

Precondition

`list` must be not NULL.
`node` must be not NULL.

Postcondition

List size increases by 1.

Remarks

Complexity: O(1)

4.4.3.2 list_advance()

```
struct list_node* list_advance (
    struct list_node * node,
    size_t distance )
```

Returns the nth-node after the current one.

Parameters

| | |
|-----------------|----------------------|
| <i>node</i> | a node. |
| <i>distance</i> | distance to move on. |

Returns

the nth-node after *node*.

Precondition

node must be not NULL.

Remarks

Complexity: O(n)

4.4.3.3 list_append()

```
void list_append (
    struct list * list,
    struct list_node * node ) [inline]
```

Adds *node* at the end of *list*.

Parameters

| | |
|-------------|---------------|
| <i>list</i> | a list. |
| <i>node</i> | the new node. |

Precondition

list must be not NULL.
node must be not NULL.

Postcondition

List size increases by 1.

Remarks

Complexity: O(n)

4.4.3.4 list_at()

```
struct list_node* list_at (
    const struct list * list,
    size_t pos )
```

Returns node at the position *pos*.

Parameters

| | |
|-------------|---------------------------------|
| <i>list</i> | a list. |
| <i>pos</i> | position (0-based) of the node. |

Returns

the node at the position *pos*.

Precondition

list must be not NULL.
list must be not empty.
pos must be in [0; list_length(list)].

Remarks

Complexity: O(N)

4.4.3.5 list_concat()

```
void list_concat (
    struct list * l1,
    struct list * l2 ) [inline]
```

Concatenates two lists.

Parameters

| | |
|-----------|---------|
| <i>l1</i> | list 1. |
| <i>l2</i> | list 2. |

Precondition

l1 must be not NULL.
l2 must be not NULL.
l1 must be different of l2.

Postcondition

l2 is reset to an empty list.

Remarks

Complexity: O(N)

4.4.3.6 list_del()

```
void list_del (
    struct list * list ) [inline]
```

Deletes the first node.

Parameters

| | |
|-------------|---------|
| <i>list</i> | a list. |
|-------------|---------|

Precondition

list must be not NULL.
list must be not empty.

Postcondition

List size decreases by 1.

Remarks

Complexity: O(1)

4.4.3.7 list_del_after()

```
void list_del_after (
    struct list * list,
    struct list_node * node ) [inline]
```

Deletes the node at after the node curr.

Parameters

| | |
|-------------|-------------------------------|
| <i>list</i> | a list. |
| <i>node</i> | a node of <code>list</code> . |

Precondition

`list` must be not NULL.
`node` must be not NULL.
`list` must be not empty.
`node` must a node of `list`.

Postcondition

List size decreases by 1.

Remarks

Complexity: O(1)

4.4.3.8 `list_del_at()`

```
void list_del_at (  
    struct list * list,  
    size_t pos ) [inline]
```

Deletes the node at the position `pos`.

Parameters

| | |
|-------------|--|
| <i>list</i> | a list. |
| <i>pos</i> | index (0-based) of the node to delete. |

Precondition

`list` must be not NULL.
`list` must be not empty.
`pos` must be in `[0; list_length(list)[`.

Postcondition

List size decreases by 1.

Remarks

Complexity: O(n)

4.4.3.9 list_first()

```
struct list_node* list_first (
    const struct list * list )
```

Returns the first node.

Parameters

| | |
|-------------|---------|
| <i>list</i> | a list. |
|-------------|---------|

Returns

the first node.

Precondition

`list` must be not NULL.
`list` must be not empty.

Remarks

Complexity: O(1)

4.4.3.10 list_init()

```
void list_init (
    struct list * list ) [inline]
```

Initializes the list.

Parameters

| | |
|-------------|---------|
| <i>list</i> | a list. |
|-------------|---------|

Precondition

`list` must be not NULL.

Postcondition

`list` is empty.
`list` has a size of 0.

Remarks

Complexity: O(1)

4.4.3.11 list_insert_after()

```
void list_insert_after (
    struct list * list,
    struct list_node * curr,
    struct list_node * node ) [inline]
```

Inserts *node* at after the node *curr*.

Parameters

| | |
|-------------|-------------------------|
| <i>list</i> | a list. |
| <i>curr</i> | a node of <i>list</i> . |
| <i>node</i> | new node. |

Precondition

list must be not NULL.
curr must be not NULL.
curr must a node of *list*.
node must be not NULL.

Postcondition

List size increases by 1.

Remarks

Complexity: O(1)

4.4.3.12 list_insert_at()

```
void list_insert_at (
    struct list * list,
    struct list_node * node,
    size_t pos ) [inline]
```

Inserts *node* at the position *pos* in *list*.

Parameters

| | |
|-------------|--|
| <i>list</i> | a list. |
| <i>node</i> | new node. |
| <i>pos</i> | position (0-based) where to insert the new node. |

Precondition

`list` must be not NULL.
`node` must be not NULL.
`pos` must be in $[0; \text{list_length}(\text{list})]$.

Postcondition

List size increases by 1.

Remarks

Complexity: $O(n)$

4.4.3.13 list_is_empty()

```
int list_is_empty (
    const struct list * list ) [inline]
```

Tests if a list is empty.

Parameters

| | |
|-------------|---------|
| <i>list</i> | a list. |
|-------------|---------|

Returns

1 if the list is empty, otherwise 0.

Precondition

`list` must be not NULL.

Remarks

Complexity: $O(1)$

4.4.3.14 list_last()

```
struct list_node* list_last (
    const struct list * list )
```

Returns the last node.

Parameters

| | |
|-------------|---------|
| <i>list</i> | a list. |
|-------------|---------|

Returns

the last node.

Precondition

`list` must be not NULL.

Remarks

Complexity: O(N)

4.4.3.15 list_length()

```
size_t list_length (  
    const struct list * list ) [inline]
```

Returns the size of the list.

Parameters

| | |
|-------------|---------|
| <i>list</i> | a list. |
|-------------|---------|

Returns

the length of the list.

Precondition

`list` must be not NULL.

Remarks

Complexity: O(1)

4.4.3.16 list_next()

```
struct list_node* list_next (  
    const struct list_node * node )
```

Returns the next node.

Parameters

| | |
|-------------|---------|
| <i>node</i> | a node. |
|-------------|---------|

Returns

the next node.

Precondition

node must be not NULL.

Remarks

Complexity: O(1)

4.4.3.17 list_print()

```
void list_print (
    struct list * list )
```

Print the list

Parameters

| | |
|-------------|--------|
| <i>list</i> | a list |
|-------------|--------|

4.4.3.18 list_reverse()

```
void list_reverse (
    struct list * list ) [inline]
```

Reverses the order of the elements in the list.

Parameters

| | |
|-------------|---------|
| <i>list</i> | a list. |
|-------------|---------|

Precondition

list must be not NULL.

Remarks

Complexity: $O(N)$

4.4.3.19 list_sort()

```
void list_sort (
    struct list * list,
    int(*) (struct list_node *, struct list_node *) cmp ) [inline]
```

Sort a list using a comparison function.

The contents of the list are sorted in ascending order according to a comparison function which is called with two arguments that point to the node being compared.

The comparison function must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second.

If two members compare as equal, their order in the sorted list is preserved.

Parameters

| | |
|-------------|-----------------------------|
| <i>list</i> | list to sort. |
| <i>cmp</i> | comparison function to use. |

Precondition

list must be not NULL.
cmp must be not NULL.

Remarks

The sort is stable.
Complexity: $O(N \log N)$
Space complexity: $O(1)$

4.4.3.20 list_split_at()

```
void list_split_at (
    struct list * list,
    size_t pos,
    struct list * right ) [inline]
```

Splits a list in two parts at the position *pos*.

After the split:

- `list` contains nodes in `[0, pos[`
- `right` contains nodes in `[pos,length(list)[`

Examples:

```
list = [1, 2, 3]
list_split_at(list, 0, right) => ([], [1,2,3])
list_split_at(list, 1, right) => ([1], [2,3])
list_split_at(list, 2, right) => ([1,2], [3])
list_split_at(list, 3, right) => ([1,2,3], [])
list = []
list_split_at(list, 0, right) => ([], [])
```

Parameters

| | |
|--------------|--|
| <i>list</i> | list to split. |
| <i>pos</i> | position (0-based) where to split the list. |
| <i>right</i> | an empty list to receive the part after <code>pos</code> |

Precondition

`list` must be not NULL.
`right` must be not NULL.
`right` must be empty.
`list` must be different of `right`.

Remarks

Complexity: O(N)

4.4.3.21 list_swap()

```
void list_swap (
    struct list * l1,
    struct list * l2 ) [inline]
```

Swaps two lists.

Parameters

| | |
|-----------|---------|
| <i>l1</i> | list 1. |
| <i>l2</i> | list 2. |

Precondition

`l1` must be not NULL.
`l2` must be not NULL.
`l1` must be different of `l2`.

Remarks

Complexity: $O(1)$

Index

- first
 - list, 5
- length
 - list, 6
- list, 5
 - first, 5
 - length, 6
- list.c
 - list_add, 9
 - list_advance, 9
 - list_append, 10
 - list_at, 10
 - list_concat, 12
 - list_del, 12
 - list_del_after, 13
 - list_del_at, 14
 - list_first, 14
 - list_init, 15
 - list_insert_after, 15
 - list_insert_at, 16
 - list_is_empty, 16
 - list_last, 17
 - list_length, 17
 - list_next, 18
 - list_print, 18
 - list_reverse, 19
 - list_sort, 19
 - list_split_at, 20
 - list_swap, 21
- list.h
 - list_add, 26
 - list_advance, 26
 - list_append, 27
 - list_at, 28
 - list_concat, 28
 - list_del, 29
 - list_del_after, 29
 - list_del_at, 30
 - list_elt, 23
 - list_first, 30
 - list_foreach, 23
 - list_foreach_elt, 24
 - list_foreach_elt_safe, 24
 - list_foreach_safe, 25
 - list_init, 31
 - list_insert_after, 31
 - list_insert_at, 32
 - list_is_empty, 33
 - list_last, 33
 - list_length, 34
 - list_next, 34
 - list_print, 35
 - list_reverse, 35
 - list_sort, 36
 - list_split_at, 36
 - list_swap, 37
- list_add
 - list.c, 9
 - list.h, 26
- list_advance
 - list.c, 9
 - list.h, 26
- list_append
 - list.c, 10
 - list.h, 27
- list_at
 - list.c, 10
 - list.h, 28
- list_concat
 - list.c, 12
 - list.h, 28
- list_del
 - list.c, 12
 - list.h, 29
- list_del_after
 - list.c, 13
 - list.h, 29
- list_del_at
 - list.c, 14
 - list.h, 30
- list_elt
 - list.h, 23
- list_first
 - list.c, 14
 - list.h, 30
- list_foreach
 - list.h, 23
- list_foreach_elt
 - list.h, 24
- list_foreach_elt_safe
 - list.h, 24
- list_foreach_safe
 - list.h, 25
- list_init
 - list.c, 15
 - list.h, 31
- list_insert_after
 - list.c, 15

- list.h, 31
- list_insert_at
 - list.c, 16
 - list.h, 32
- list_is_empty
 - list.c, 16
 - list.h, 33
- list_last
 - list.c, 17
 - list.h, 33
- list_length
 - list.c, 17
 - list.h, 34
- list_next
 - list.c, 18
 - list.h, 34
- list_node, 6
 - next, 6
- list_print
 - list.c, 18
 - list.h, 35
- list_reverse
 - list.c, 19
 - list.h, 35
- list_sort
 - list.c, 19
 - list.h, 36
- list_split_at
 - list.c, 20
 - list.h, 36
- list_swap
 - list.c, 21
 - list.h, 37
- main
 - tAltris.c, 7
- next
 - list_node, 6
- src/tAltris.c, 7
- src/tAltris.h, 8
- src/utlis/list.c, 8
- src/utlis/list.h, 21
- tAltris.c
 - main, 7