libncc Documentation

Table of Contents

1 Description

1.1 About

libncc is a static C library which provides data structures and operations to handle:

- lists
- stacks
- queues

It is based on the four LISPs' list functions:

- null
- car
- cons
- cdr

1.2 Internals

1.2.1 Directories

- `/src/private` contains basic operations and CDT (Concrete Data Types).
- `/src/adt` contains functions that provide higher lever operations. These operations are called ADT (Abstract Data Types).
- `/src/public` contains operations visible to the end user. These operations are called API (Application Programming Interface).

1.2.2 Test modules

All test modules are collected in the `/src/test.c` file. You can run all the test modules with `cd src && ./runTests.sh` or by calling the `src/Makefile` targets manually.

Whenever a test module is compiled, this is automatically checked for errors thanks to the `src/validate.sh` script.

To auto-indent all C files, you can simply `cd src && make indent`.

1.2.3 C Flags

This library is written in ANSI C99. Compilation flags follow:

```
-g -Wall -Wextra -Wpedantic -Werror -march=native -O0 -lrt -std=c99
```

1.2.4 Indent Flags

GNU indentation options have been used:

```
-nbad -bap -nbc -bbo -bl -bli2 -bls -ncdb -nce -cp1 -cs -di2 -ndj -nfc1-nfca
-hnl -i2 -ip5 -lp -pcs -psl -nsc -nsob
```

2 API Description

Everything described in this chapter can be found in `/include/libncc.h`.

2.1 Variable Definitions

`_node` *list* [typedef]

`_node` *stack* [typedef]

`_node` *queue* [typedef]

2.2 Function Descriptions

2.2.1 Functional operations

`bool list_null` (*list l*) [List]
Check if a list is `NULL`.

`element list_head` (*list l*) [List]
Extracts the first element of the list.

`list list_next` (*list l*) [List]
Gets the pointer of the next node of a list.

`int list_length` (*list l*) [List]
Returns the length of a list.

`bool list_same` (*list l1*, *list l2*) [List,Stack,Queue]
Checks if the `element` part of two `_node` objects are equal.

`bool list_equal` (*list l1*, *list l2*) [List,Stack,Queue]
Checks if the two `_node` object sets are equal.

`bool stack_null` (*stack s*) [Stack]
Checks if a stack is `NULL`.

`int stack_length` (*stack s*) [Stack]
Computes the number of elements in the stack.

`bool queue_null` (*queue q*) [Queue]
Checks if a queue is `NULL`.

`int queue_length` (*queue q*) [Queue]
Computes the number of elements in the queue.

2.2.2 Non-functional operations

`void list_init` (*list * `lRef`*) [List]
Sets the input list to `NULL`.

`void list_append` (*element e, list * `lRef`*) [List]
Adds an element to the tail of the list.

`void list_prepend` (*element e, list * `lRef`*) [List]
Adds an element to the head of the list.

`void list_remove` (*list * `head`, list `toRemove`*) [List,Stack,Queue]
Removes a specified element from a `_node` object set.

`void list_destroy` (*list * `lRef`*) [List,Stack,Queue]
Destroy a `_node` object set from the specified statring point.

`void stack_init` (*stack * `sRef`*) [Stack]
Sets the input stack to `NULL`.

`element stack_pop` (*stack * `sRef`*) [Stack]
Gets the first element and frees its corresponding `_node` object of the stack.

`void stack_push` (*element `e`, stack * `sRef`*) [Stack]
Inserts a new element in the the stack.

`void queue_init` (*queue * `qRef`*) [Queue]
Sets the input queue to `NULL`.

`element queue_dequeue` (*queue * `qRef`*) [Queue]
Removes the tail element from the queue.

`void queue_enqueue` (*element `e`, queue * `qRef`*) [Queue]
Adds an element from the queue.

3 Usage

3.1 Example

```
/*
 * example.c
 *
 * Copyright copyright 2016 Franco Masotti <franco.masotti@student.unife.it>
 * This work is free.  You can redistribute it and/or modify it under the
 * terms of the Do What The Fuck You Want To Public License, Version 2,
 * as published by Sam Hocevar.  See the LICENSE file for more details.
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "libncc.h"

int main (void)
{
    int n = 5;
    int i;
    int *arr = malloc (sizeof (int) * n);
    list l;
    stack s;
    queue q;

    list_init (&l);
    stack_init (&s);
    queue_init (&q);

    fprintf (stderr, "l, s, q\n");

    for (i = 0; i < n; i++)
    {
        arr[i] = i;

        /* You can test both append and prepend functions here.  */
        list_append (arr + i, &l);
        stack_push (arr + i, &s);
        queue_enqueue (arr + i, &q);
        fprintf (stderr, "Lengths = %d, %d, %d\n", list_length (l),
list_length (s), list_length (q));
    }

    while (!list_null (l) || !stack_null (s) || !queue_null (q))
```

```
    {
        fprintf (stderr, "%d ", *(list_head (l)));
        list_remove (&l, l);
        fprintf (stderr, "%d ", *(stack_pop (&s)));
        fprintf (stderr, "%d ", *(queue_dequeue (&q)));
        fprintf (stderr, "\n");
    }

    fprintf (stderr, "\n");

    free (arr);

    return 0;
}
```

3.2 Building and Linking

3.2.1 Building

To build the static library into the `libncc.a` file you should use the `Makefile`

`make libncc TYPE=`*`typedef struct or native C type`*

For example if you want to use `int *` as elements your command should be:

`make libncc TYPE=`*`int`*

This means that you cannot pass explicit pointers.

Another example: if you want to link `struct` *`something`* `*`, somewhere in you code, you should have `typedef struct` *`something smt`*`;`

You can then run:

`make libncc TYPE=`*`smt`*

3.2.2 Linking

To link the library you can either run the following command or modify the `Makefile` directly. A possible command could be:

`gcc -o example.out example.c libncc.a`