

libncc Documentation

Copyright © 2016 Franco Masotti `franco.masotti@student.unife.it`

Copying and distribution of this file, with or without modification, are permitted in any medium without royalty provided the copyright notice and this notice are preserved.

Table of Contents

1	Description.....	1
1.1	About.....	1
1.2	Internals.....	1
1.2.1	Directories.....	1
1.2.2	Test modules.....	1
1.2.3	C Flags.....	1
1.2.4	Indent Flags.....	1
2	API Description.....	2
2.1	Variable Definitions.....	2
2.2	Function Descriptions.....	2
2.2.1	Functional operations.....	2
2.2.2	Non-functional operations.....	3
3	Usage.....	4
3.1	Example.....	4
3.2	Building and Linking.....	5
3.2.1	Building.....	5
3.2.2	Linking.....	6

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 using the ISO C99 standard. 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

<code>_node list</code>	[typedef]
<code>_node stack</code>	[typedef]
<code>_node queue</code>	[typedef]

2.2 Function Descriptions

2.2.1 Functional operations

<code>bool list_null (list l)</code>	[List]
Check if a list is NULL.	
<code>element list_head (list l)</code>	[List]
Extracts the first element of the list.	
<code>list list_next (list l)</code>	[List]
Gets the pointer of the next node of a list.	
<code>int list_length (list l)</code>	[List]
Returns the length of a list.	
<code>bool list_same (list l1, list l2)</code>	[List,Stack,Queue]
Checks if the <code>element</code> part of two <code>_node</code> objects are equal.	
<code>bool list_equal (list l1, list l2)</code>	[List,Stack,Queue]
Checks if the two <code>_node</code> object sets are equal.	
<code>bool stack_null (stack s)</code>	[Stack]
Checks if a stack is NULL.	
<code>int stack_length (stack s)</code>	[Stack]
Computes the number of elements in the stack.	
<code>bool queue_null (queue q)</code>	[Queue]
Checks if a queue is NULL.	
<code>int queue_length (queue q)</code>	[Queue]
Computes the number of elements in the queue.	

2.2.2 Non-functional operations

void <code>list_init</code> (<i>list</i> * <i>lRef</i>)	[List]
Sets the input list to NULL.	
void <code>list_append</code> (<i>element</i> <i>e</i> , <i>list</i> * <i>lRef</i>)	[List]
Adds an element to the tail of the list.	
void <code>list_prepend</code> (<i>element</i> <i>e</i> , <i>list</i> * <i>lRef</i>)	[List]
Adds an element to the head of the list.	
void <code>list_remove</code> (<i>list</i> * <i>head</i> , <i>list toRemove</i>)	[List,Stack,Queue]
Removes a specified element from a <code>_node</code> object set.	
void <code>list_destroy</code> (<i>list</i> * <i>lRef</i>)	[List,Stack,Queue]
Destroy a <code>_node</code> object set from the specified starting point.	
void <code>stack_init</code> (<i>stack</i> * <i>sRef</i>)	[Stack]
Sets the input stack to NULL.	
element <code>stack_pop</code> (<i>stack</i> * <i>sRef</i>)	[Stack]
Gets the first element and frees its corresponding <code>_node</code> object of the stack.	
void <code>stack_push</code> (<i>element</i> <i>e</i> , <i>stack</i> * <i>sRef</i>)	[Stack]
Inserts a new element in the the stack.	
void <code>queue_init</code> (<i>queue</i> * <i>qRef</i>)	[Queue]
Sets the input queue to NULL.	
element <code>queue_dequeue</code> (<i>queue</i> * <i>qRef</i>)	[Queue]
Removes the tail element from the queue.	
void <code>queue_enqueue</code> (<i>element</i> <i>e</i> , <i>queue</i> * <i>qRef</i>)	[Queue]
Adds an element from the queue.	

3 Usage

3.1 Example

```

/*
 * example.h
 *
 * 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.
 */

#if !defined EXAMPLE_H
#define EXAMPLE_H

struct example
{
    int num;
    char *aChar;
};

typedef struct example ex;

#endif

/*
 * 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"
#include "example.h"

int main (void)
{
    int n = 5;
    int i;
    ex *arr = malloc (sizeof (struct example) * n);
    list l;

```

```

    stack s;
    queue q;
    char h[6] = {'h', 'a', 'l', 'l', 'o', '\0'};

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

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

    for (i = 0; i < n; i++)
    {
        arr[i].aChar = malloc (sizeof (char) * n);
        sprintf (arr[i].aChar, "%s:%d", h, 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, "%s ", (*(list_head (l))).aChar);
        list_remove (&l, l);
        fprintf (stderr, "%s ", (*(stack_pop (&s))).aChar);
        fprintf (stderr, "%s ", (*(queue_dequeue (&q))).aChar);
        fprintf (stderr, "\n");
    }

    fprintf (stderr, "\n");

    for (i = 0; i < n; i++)
        free (arr[i].aChar);
    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=native C type
```


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

```
make libncc TYPE=int
```

Another example: if you want to link `struct something *`, somewhere in you code, you should have a definition of that structure

```
struct something { ... };
```

However, you cannot run the following because the structure definition remains unknown:

```
make libncc TYPE="struct\ something"
```

Anyway, you can use structures in the linking phase.

3.2.2 Linking

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

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

As an example, you can run `make example`, which uses the `TYPE="struct\ example"` option.

```
#!/usr/bin/make -f
```

```
#
# Makefile
#
# 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.
#
```

```
# Structure name.
TYPE=struct\ example
CC = /usr/bin/gcc
INCLUDE = -I../include
# Library include directory path.
INCLUDELIB = -I../include
CFLAGS = -g -Wall -Wextra -Wpedantic -Werror -march=native -O0
LIBS = -lrt
CSTANDARD = -std=c99
DEFFLAG = -DelementObject=$(TYPE)
```

```
default: example
```

```
.PHONY: default libncc example example.out
```

```
clean:
    @echo "Removing object files..."
    @rm -fv *.o *.out *.a
    @echo "Object files removed."
```

```

%.o: %.c
    @$(CC) $(INCLUDE) $(INCLUDELIB) $(CFLAGS) $(CSTANDARD) \
$(DEFFLAG) $(LIBS) -c -o $@ $<

libncc:
    @$(MAKE) -C ../../
    @mv ../../$@.a .

example: example.o libncc
    @$(CC) $(INCLUDE) $(INCLUDELIB) $(CFLAGS) $(CSTANDARD) \
$(DEFFLAG) $(LIBS) -o $@.out example.o libncc.a
    @echo "$(CC) $(INCLUDE) $(INCLUDELIB) $(CFLAGS) $(CSTANDARD) \
$(DEFFLAG) $(LIBS) -o $@.out example.o libncc.a"

```

This is possible thanks to opaque structure definition. It only works if you compile and link the library and you program simultaneously¹.

If you want to do a simpler linking, you must edit the library directly, just like what has been done for the test modules. For more information see `/include/list_base.h` and `/include/test_struct.h`.

¹ Due to visibility problems you must put the `libncc.a` file directive at the end of the target, as shown in the example.