Exercises — Dynamic Queue

version #

Assistants C/Unix 2022 <assistants@tickets.assistants.epita.fr>

Copyright

This document is for internal use at EPITA (website) only.

Contents

*https://intra.assistants.epita.fr

1 Dynamic Queue

Files to submit:

- fifo/Makefile
- fifo/*

Provided files:

- fifo/fifo.h

Makefile: Your makefile should define at least the following targets:

- library: Produces the libfifo.a library containing all the following functions at the root of the fifo exercise. This must be the first target.
- clean: Deletes everything produced by make

Authorized functions: You are only allowed to use the following functions:

- calloc(3)
- free(3)
- malloc(3)
- printf(3)
- putchar(3)
- puts(3)

Authorized headers: You are only allowed to use the functions defined in the following headers:

- stddef.h
- err.h
- assert.h
- errno.h

1.1 Goal

The goal here is to achieve a functional queue.

Fifo (First in first out), also know as queue, is a data structure to organized your data, that act like a waiting list: elements goes out of the structure in the same order that they were inserted.

You are free to name your files however you like. As long as the coding style is respected.

1.2 Preamble

You must use the file `fifo.h` that is given to you.

1.2.1 Description

Your structure holding the elements must be named `list`:

```
struct list
{
    int data;
    struct list *next;
};
```

The queue structure must be named `fifo`:

```
struct fifo
{
    struct list *head;
    struct list *tail;
    size_t size;
};
```

- You must use the above structures as is for your grade to be positive.
- The case where the pointer to the queue is `NULL` will not be tested.
- If `malloc` fails: the function must not behave as expected (e.g. no modification like size increase). If the function should have returned a pointer, `NULL` must be returned instead.
- Note that the type in the `fifo` structure is `size_t`. This type can be found in the `stddef.h` header file.

1.3 fifo_init

```
struct fifo *fifo_init(void);
```

Create a new queue and initialize it.

1.4 fifo_size

```
size_t fifo_size(struct fifo *fifo);
```

Return the number of elements in the queue.

1.5 fifo_push

```
void fifo_push(struct fifo *fifo, int elt);
```

Add a new element to the queue.

1.6 fifo_head

```
int fifo_head(struct fifo *fifo);
```

Return the head element of the queue (i.e. the next to be removed). The case when the queue is empty will not be tested.

1.7 fifo_pop

```
void fifo_pop(struct fifo *fifo);
```

Remove the head of the queue. If the queue is empty, nothing has to be done. Take care to not loose the tail of the queue and to not cause memory leaks.

1.8 fifo_clear

```
void fifo_clear(struct fifo *fifo);
```

Remove all elements of the queue. Your queue must be usable after the call to clear. Once again, take care of your memory management.

1.9 fifo_destroy

```
void fifo_destroy(struct fifo *fifo);
```

Empty the queue and free the `fifo` structure. Free all resources.

1.10 fifo_print

```
void fifo_print(const struct fifo *fifo);
```

Display the elements of the list from head to tail. A line feed must delimit each element. A line feed must also be printed after the last element. If the list is empty, nothing must be printed.

```
42sh$ ./fifo_print | cat -e
1$
-2$
3$
42sh$ ./fifo_print_empty | cat -e
42sh$
```

You must display the queue from head to tail.

It is my job to make sure you do yours.