Exercises — Heap

version #

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

Copyright

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

Contents

1 Heap **3**
- 1.1 Goal . . . 3
- 1.2 Create . . . 4
- 1.3 Add . . . 4
- 1.4 Pop . . . 4
- 1.5 Delete . . . 5
- 1.6 Print . . . 5

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

1 Heap

Files to submit:

- heap/Makefile
- heap/create.c
- heap/add.c
- heap/pop.c
- heap/del.c
- heap/print.c
- heap/heap.h

Provided files:

- heap/heap.h

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

- library: Produce the libheap.a library
- clean: Delete everything produced by make

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

- malloc(3)
- realloc(3)
- free(3)
- putchar(3)
- assert(3)

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

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

1.1 Goal

You might have seen an implementation of heaps using pointers. Here, you are going to re-implement a heap, but using a vector. This heap is a binary max-heap.

As a reminder, remember that for a node at position i, its left child will be at position $2i + 1$ and its right one at position $2i + 2$.

The structure used for this exercise is the following:

```c
struct heap
{
    size_t size;
    size_t capacity;
    int *array;
};
```

The cases where heap is `NULL` will not be tested.

For this exercise, we provide you a `heap.h` file. You do not have to edit this file: during the tests it will be overwritten anyway.

Notes:

- Manage the memory wisely.
- Test your code and always check tricky input instances.

1.2 Create

- **Filename:** `create.c`

Write the following function:

```c
struct heap *create_heap(void);
```

This function initializes the heap. It returns an allocated heap with a `size` initialized to 0, and an `array` with a capacity of 8.

1.3 Add

- **Filename:** `add.c`

Write the following function:

```c
void add(struct heap *heap, int val);
```

This function adds a new value to the heap by creating a new slot in it. If the heap is full, `add` will automatically expand it.

1.4 Pop

- **Filename:** `pop.c`

Write the following function:

```c
int pop(struct heap *heap);
```

This function returns the root of the heap and deletes it. If the heap is empty, the program stops with an `Assertion Failed`.

1.5 Delete

- **Filename:** `del.c`

Write the following function:

```
void delete_heap(struct heap *heap);
```

This function removes the heap and all its elements. After it has been called, the heap will not be usable anymore.

1.6 Print

- **Filename:** `print.c`

Write the following function:

```
void print_heap(const struct heap *heap);
```

This function displays on the standard output all the elements of the heap with a pre-order depth first traversal, separated by spaces, ending with a newline character (`'\n'`). The output format is:

```
42sh$ ./print_heap | cat -e
e1 e2 e3 eN$
```

with `e1` ... `eN` the values of the nodes.

It is my job to make sure you do yours.