Exercises — Heap

version #7be580532266ed398481e31366afcc24b1950c2a

The way is lit. The path is clear.
We require only the strength to follow it.

Assistants C/Unix 2023 <assistants@tickets.assistants.epita.fr>

Copyright

Contents

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

File Tree

```
heap/
├─ Makefile  (to submit)
├─ add.c  (to submit)
├─ create.c  (to submit)
├─ del.c  (to submit)
├─ heap.h  (to submit)
├─ pop.c  (to submit)
└─ print.c  (to submit)
```

Makefile

- 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)
- printf(3)

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

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

Compilation : Your code must compile with the following flags

- -std=c99 -pedantic -Werror -Wall -Wextra -Wvla

Main function : None

1 Goal

In computer science, a heap is a specialized tree-based data structure which is essentially an almost complete tree that satisfies the heap property: in a max heap, for any given node C, if P is a parent node of C, then the key (the value) of P is greater than or equal to the key of C.

Here, you are going to implement a heap 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:

```
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.

2 Create

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

Write the following function:

```
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. If any error occurs, you should return NULL.

3 Add

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

Write the following function:

```
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 double its capacity.

4 Pop

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

Write the following function:

```
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`. If the heap's size is lower than half its capacity `pop` will automatically reduce its capacity by half, but never less than 8.

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.

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.

The way is lit. The path is clear. We require only the strength to follow it.