

## CSE310 Project 2: Min-Heap

Posted: 03/02/2021, Due: 03/15/2021

In your second programming project, you **will expand your first project**. As in the case of the first programming project, it should be implemented in C++, on a Linux platform such as `general.asu.edu`. Your program will be graded on Gradescope, which uses a Linux platform. You will continue to have modular design, provide a **Makefile** to compile various modules to generate the executable file named `run`. Among other things, you need to have

1. a main program, which coordinates all other modules;
2. a module that provides utility services including command line interpretation;
3. a module that implements the min-heap data structure (**all heap functions**);
4. a Makefile which compiles all modules and link them into the executable.

For each module other than the main program, you should have a **header file** which specifies the data structures and the prototypes of the functions in the module, and an **implementation file** which implements all of the functions specified in the header file. **All programs will be compiled and graded on Gradescope**. If your program works well on `general.asu.edu`, there should not be much problems. You will need to submit it electronically on Gradescope via the link on Canvas. Test cases are posted on Canvas. **If your program does not compile and work on Gradescope, you will receive 0 on this project.**

You need to define the following data types.

- `ELEMENT` is a `struct` that contains a field named `key`, which is of type `int`.
- `HEAP` is a data type that contains three fields named `capacity` (of type `int`), `size` (of type `int`), and `H` (of type `**ELEMENT`). `H` will be pointing to an array of `capacity + 1` of pointers of type `*ELEMENT`. Note that the size of `HEAP` should be equal to 12, regardless of the `capacity` or the `size` of the heap. In other words, **`sizeof(HEAP)` should always return 12.**

The functions that you are required to implement are:

- `Initialize(value)` which creates an object of type `HEAP` with `capacity` equal to `value`, `size` equal to 0, and `H` points to a dynamically allocated array of `value + 1` pointers. It then **returns a pointer to this object**. This function requires you to perform dynamic memory allocation, given the demand `value`. Note that you have already done this in your first project.
- `PrintHeap(heap)` which prints out the information of the heap pointed to by `heap`, including `capacity`, `size`, and the `key` fields of the elements in the array with index going from 1 to `size`. Note that you have already done this in your first project.
- `WriteHeap(heap)` which opens the file named "HEAPoutput.txt" in write mode and write the the information of the heap pointed to by `heap` to the file. The format of "HEAPoutput.txt" should be identical to that of "HEAPinput.txt".

- `Insert(heap, element)` which inserts an object of type `ELEMENT` pointed to by `element` into the heap pointed to by `heap`.
- `ExtractMin(heap)` which deletes the minimum element from the heap pointed to by `heap`.
- `DecreaseKey(heap, index, value)` which decreases the key of `heap->H[index]` to `value`.
- `min-Heapify(heap, index)` which performs the heapify function at `index` of the array pointed to by `heap->H`.

You are free to include additional parameters as you see fit and decide the return types of the functions. You should implement a module that takes the following commands from `stdin` and feeds to the main program:

- `S`
- `P`
- `W`
- `C value`
- `R flag`
- `I value`
- `D flag`
- `K index value`

The main program should react to each of the above commands in the following way.

S: On reading `S`, the program

1. Stops.

P: On reading `P`, the program does the following:

1. If `heap` is `NULL`, writes the following line to `stdout`:  
`Error: cannot print`  
and waits for the next command from `stdin`.
2. Writes the information of the heap pointed to by `heap` to `stdout`. [Refer to the posted test cases for output format.](#)
3. Waits for the next command from `stdin`.

W: On reading `W`, the program does the following:

1. Opens the file "HEAPout.txt" in write mode. If the file is not opened successfully, or `heap` is `NULL`, writes the following line to `stdout`:

Error: cannot write

and waits for the next command from `stdin`.

2. Writes the information of the heap pointed to by `heap` to the file "HEAPoutput.txt". "HEAPoutput.txt" should have exactly the same format as "HEAPinput.txt".
3. Waits for the next command from `stdin`.

C: On reading C value, the program does the following:

1. Calls a function in the `heap` module to create a heap with `capacity` equal to `value` and `size` equal to 0, and return a pointer to this heap object to the caller.
2. Waits for the next command from `stdin`.

R: On reading R flag, the program does the following:

1. Opens the file "HEAPinput.txt" in read mode. If the file is not opened successfully, writes the following line to `stdout`:

Error: cannot open file for reading

and waits for the next command from `stdin`.

2. Reads in the first integer, `n`, from the file opened. If `heap` is NULL or `heap->capacity` is smaller than `n`, writes the following line to `stdout`:

Error: heap overflow

and waits for the next command from `stdin`.

3. Reads in the next  $n$  integers  $key_1, key_2, \dots, key_n$  from the file, dynamically allocates memory for an `ELEMENT`, sets its `key` to  $key_j$ , and let `heap->H[j]` points to this `ELEMENT`, for  $j = 1, 2, \dots, n$ .
4. Calls the `BuildHeap` within the `heap` module to build a min heap on the array `H` using the linear time `buildheap` algorithm. **Note that you have to change the algorithm accordingly to build a min-heap, not a max-heap.**
5. If `flag` is equal to 1, your program should write to `stdout` the number of `min-Heapify` functions called by this `BuildHeap` call. [Refer to the test cases for the output format.](#)
6. Waits for the next command from `stdin`.

D: On reading D flag, the program does the following:

1. If `heap` is NULL or `heap->size` is 0, writes the following line to `stdout`:

Error: heap is NULL or empty

and waits for the next command from `stdin`.

2. Deletes the minimum element from the heap and writes to `stdout` the key of the deleted element. [Refer to the test cases for output format.](#)

3. If `flag` is equal to 1, your program should write to `stdout` the number of `min-Heapify` functions triggered by this `delete-min` operation. Refer to the test cases for the output format.
4. Waits for the next command from `stdin`.

I: On reading I value, the program does the following:

1. If `heap` is `NULL` or `heap->size` is equal to `heap->capacity`, writes the following line to `stdout`:  
 Error: heap is NULL or full  
 and waits for the next command from `stdin`.
2. Dynamically allocates memory for an `ELEMENT`, sets its `key` field to `value`, and inserts it to the heap pointed to by `heap`.
3. Waits for the next command from `stdin`.

K: On reading K index value, the program does the following:

1. If `heap` is `NULL` or `index` is not within the interval `[1, heap->size]`, or `value` is greater than or equal to `heap->H[index]->key`, writes the following line to `stdout`:  
 Error: invalid call to DecreaseKey  
 and waits for the next command from `stdin`.
2. Perform the corresponding decrease key operation: decrease the key of `heap->H[index]` to `value` by calling the `DecreaseKey` function in the `heap` module.
3. Waits for the next command from `stdin`.

The file `HEAPinput.txt` is a text file. The first line of the file contains an integer  $n$ , which indicates the number of array elements. The next  $n$  lines contain  $n$  integers, one integer per line. These integers are the key values of the  $n$  array elements, from the first element to the  $n$ th element. Refer to the posted test cases for the exact format of "HEAPinput.txt".

**Grading policies:** (Sample test cases are posted on Canvas.) All programs will be graded on Gradescope. If your program does not compile and execute on Gradescope, you will receive 0 for this project. So start working today, and do not claim "my program works perfectly on my PC, but I do not know how to use Gradescope."

(0 pt) You should provide a `Makefile` that can be used to compile your project on Gradescope. The executable file should be named `run`. If your program does not pass this step, you will receive 0 on this project.

(60 pts) You will earn 5 points for each of the 12 posted test cases your program passes on Gradescope.

(20 pts) You will earn 5 points for each of the 4 unposted test cases your program passes on Gradescope.

You should try to make your program as robust as possible. A basic principle is that your program can complain about bad input, but should not crash.