CSE310 Project 2: Priority Queues
Due: 10/15/2023, Posted: 9/25/2023.
Check Point 1: 10/1/2023
Check Point 2: 10/8/2023

This should be your individual work. While you can use the internet to aid your work, you should write your own code. As in the case of your first programming project, it should be written using the standard `C++` programming language, and compiled using the `g++` compiler on a Linux platform. Your project will be graded on Gradescope. If you compile your project on `general.asu.edu` using the compiler commands provided in the sample `Makefile` for the first project, you should expect the same behavior of your project on both `general.asu.edu` and Gradescope. Therefore, you are highly recommended to develop your project on `general.asu.edu`.

For Project 1, I provided you with many help files for you to learn the right modular design style, as well as many examples for memory management. For Project 2, no help files are provided. It is assumed that you understand everything in the help files provided to you in Project 1, and can modify them for use in this project.

1 Data Structures and Functions

In class, we studied the **max heap** data structure and the basic max heap functions `Heapify`, `BuildHeap`, `ExtractMax`, `IncreaseKey`, and `Insertion`. Symmetrically, we have the **min heap** data structure (in which the key value at a node cannot be smaller than the value at its parent) and the corresponding basic min heap functions `Heapify`, `BuildHeap`, `ExtractMin`, `DecreaseKey`, and `Insertion`. In this project, you will implement the **min heap** data structure.

In class, for ease of presentation, we presented the functions for **max heap** under the assumption that `A[i]` is of type `int` for each valid index `i`. In this project, the type of `A[i]` should be either `ELEMENT *` or `ELEMENT`, depending on your implementation, where `ELEMENT` is a `struct` that contains a field named `key` of type `double`, along with other fields. Therefore, the heap order is decided by `A[i]->key` or `A[i].key`, depending on your implementation preference. In this document, I will assume that the type of `A[i]` is `ELEMENT *`.

1.1 Data Types

The project description will make reference to the following two data types. You may use them with modifications, but you do not have to use them.

```
typedef struct TAG_ELEMENT{
            double key;
            other fields as you see fit
}ELEMENT;

typedef struct TAG_HEAP{
            int capacity;  /* capacity of the heap */
            int size;      /* current size of the heap */
            ELEMENT **H;   /* array of pointers to ELEMENT */
            other fields as you see fit
}HEAP;
```

2 Modular Design

You will continue to do modular design, provide a `Makefile` to compile various modules to generate the executable file named `PJ2`. Among other things, you need to have at least the following modules:

1. `main.cpp`, which coordinates all modules;
2. `util.h` and `util.cpp`, which provides utility services including command line interpretation;
3. `heap.h` and `heap.cpp`, which implements the functions for min heap.

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. It is recommended that you define all data types in a header file named `data_structures.h`, and specify the prototypes of all heap functions in the header file named `heap.h`.

3 Flow of the Project

3.1 Valid Execution

A valid execution of your project has the following form:

```
./PJ2 <FileI> <FileO> flag
```

where `./PJ2` tells the system to search for the executable file `PJ2` in the current directory, `<FileI>` is the input file, `<FileO>` is the output file, and `flag` is an integer in $\{0, 1, 2, 3\}$. When the value of `flag` is 1 or 3, your program counts the total number of `Heapify` calls for each `BuildHeap` or `ExtractMin` operation. When the value of `flag` is not 1 or 3, your program does not do the above. When the value of `flag` is 2 or 3, your program prints out the location of the newly inserted element. When the value of `flag` is not 2 or 3, your program does not do the above.

Your program should check whether the execution is valid. If the execution is not valid, your program should print out the following message to `stderr` and stop.

```
Usage: ./PJ2 <FileI> <FileO> flag
<FileI>    is the input file
<FileO>    is the input file
flag       is in {0, 1, 2, 3}
```

Note that your program should not crash when the execution is not valid.

Upon a valid execution, your program should expect the following instructions from `stdin` and act accordingly:

- `Stop`:
  On reading `Stop`, the program should print the following to `stdout`

  ```
  Instruction: Stop
  ```

  (one line with the newline character at the end) and stop.

- `Init <Capacity>`:
  On reading `Init <Capacity>`, the program should do the following.

  1. Print the following line

     ```
     Instruction: Init <Capacity>
     ```

     (<Capacity> printed using "%d" format) to `stdout`.

  2. Allocate memory for an object of type `HEAP`, and return a pointer to this object. Please note that your program should also allocate memory for an heap array of size `<Capacity>`. You should also set the `size` of this `HEAP` object to 0.

  3. Wait for the next instruction from `stdin`.

Your program should return a null pointer when either of the memory allocation in the above fails. An error message should be written to `stderr` when a memory allocation fails. If the first memory allocation fails, your program does not perform the second memory allocation. If the second memory allocation fails, your program should free the object allocated in the first memory allocation.

- `Print`:
  On reading `Print`, the program should do the following.

  1. Print the following line

     `Instruction: Print`

     to `stdout`.
  2. If the heap is `NULL`, print the following error message to `stderr`:

     `Error: heap is NUll`

     then wait for the next instruction from `stdin`, skipping the following action(s).
  3. If the heap is not `NULL`, print the current state of the heap to `stdout` then wait for the next instruction from `stdin`.

  When the heap is not `NULL`, the program writes to `stdout` the `size` of the heap (not the `capacity`), followed by the key values of the elements in the heap (refer to posted test cases for output format).

- `Write`:
  On reading `Write`, the program should do the following.

  1. Print the following line

     `Instruction: Write`

     to `stdout`.
  2. If the heap is `NULL`, print the following error message to `stderr`:

     `Error: heap is NULL`

     then wait for the next instruction from `stdin`, skipping the following action(s).
  3. Opens the file `argv[2]` in write mode.
     If the file is not opened successfully, write an error message to `stderr` and wait for the next instruction from `stdin`, skipping the following actions.

4. Write the heap information to the file `argv[2]` (refer to posted test cases for format). Close the file `argv[2]`.
5. Wait for the next instruction from `stdin`.

- `Read`:
  On reading `Read`, the program should do the following.

  1. Print the following line

     ```
     Instruction: Read
     ```

     to `stdout`.
  2. If the heap is `NULL`, print the following error message to `stderr`:

     ```
     Error: heap is NULL
     ```

     then wait for the next instruction from `stdin`, skipping the following action(s).
  3. Open the file `argv[1]` in read mode.
     If the file is not opened successfully, write an error message to `stderr` and wait for the next instruction from `stdin`, skipping the following actions.
  4. Read in the first integer, `n`, from the file opened.
     If `heap->capacity` is smaller than `n`, write an error message to `stderr`, close the file `argv[1]`, and wait for the next instruction from `stdin`, skip the following actions.
  5. For each value of $j = 1, 2, \ldots, n$, read in key_j from the input file; dynamically allocate memory for an object of type `ELEMENT`; sets the `key` field of this object to key_j; let the j-th element of the corresponding array in the heap point to this object. Close the file `argv[1]`.
  6. If the input array does NOT satisy the heap order, i.e., the input array is not a MinHeap, apply the $O(n)$ time `BuildHeap` function to build the min heap. If `flag` is 1 or 3, write the following three lines to `stdout`:

     ```
     Input array is NOT a MinHeap
     Call BuildHeap
     Number of Heapify calls triggered by BuildHeap: <count>
     ```

     where <`count`> is the number of Heapify calls, using the "%d" format.
  7. Wait for the next instruction from `stdin`.

- `Insert <Key>`:
  On reading `Insert <Key>`, the program should do the following.

  1. Print the following line

     `Instruction: Insert <key>`

     to `stdout`, where <`key`> is printed using the "%lf" format.

  2. If the heap is `NULL`, print the following error message to `stderr`:

     `Error: heap is NULL`

     then wait for the next instruction from `stdin`, skipping the following actions.

  3. If the heap is full, print the following error message to `stderr`:

     `Error: heap is full`

     then wait for the next instruction from `stdin`, skipping the following actions.

  4. Dynamically allocate memory for an object of type `ELEMENT`. Set the `key` field of this object to `<Key>`. Insert this object to the min heap. If `flag` is 2 or 3, write the following line to `stdout`:

     `Element with key <key> inserted at location <pos> of the heap array`

     where <`key`> is printed using the "%lf" format and <`pos`> (the index of the heap array where the element is inserted) is printed using the "%d" format. If `flag` is 0 or 1, write the following line to `stdout`:

     `Element with key <key> inserted into the heap`

     where <`key`> is printed using the "%lf" format.

  5. Wait for the next instruction from `stdin`.

- `ExtractMin`:
  On reading `ExtractMin`, the program should do the following.

  1. Print the following line

     `Instruction: ExtractMin`

     to `stdout`.

  2. If the heap is `NULL`, print the following error message to `stderr`:

`Error: heap is NULL`

then wait for the next instruction from `stdin`, skipping the following actions.

3. If the heap is empty, print the following error message to `stderr`:

   `Error: heap is empty`

   then wait for the next instruction from `stdin`, skipping the following actions.

4. Perform the `ExtractMin` operation. If `flag` is 1 or 3, write the following two lines to `stdout`:

   `Element with key <key> extracted from the heap`
   `Number of Heapify calls triggered by ExtractMin: <count>`

   where <`key`> is printed using the "%lf" format and <`count`> (the number of Heapify calls triggered by ExtractMin) is printed using the "%d" format. If `flag` is 0 or 2, write the following line to `stdout`:

   `Element with key <key> extracted from the heap`

   where <`key`> is printed using the "%lf" format.

5. Wait for the next instruction from `stdin`.

- `DecreaseKey <Postion> <NewKey>`:
  On reading `DecreaseKey <Position> <NewKey>`, the program should do the following. Here `<Position>` is the index to the array for the **min heap** in the given data structure, and `<NewKey>` is the new value of the `key` field of the object pointed to by the corresponding array at index `<Position>`.

  1. Print the following line

     `Instruction: DecreaseKey <Position> <NewKey>`

     to `stdout`, where `int` is printed using the "%d" format and `double` is printed using the "%lf" format.

  2. If the heap is `NULL`, print the following error message to `stderr`:

     `Error: heap is NULL`

     then wait for the next instruction from `stdin`, skipping the following actions.

  3. If the heap is empty, print the following error message to `stderr`:

`Error: heap is empty`

then wait for the next instruction from `stdin`, skipping the following actions.

4. If `<Position>` if out of range or `<NewKey>` is not smaller than the corrent key, print the following error message to `stderr`:

   `Error: invalid call to DecreaseKey`

   then wait for the next instruction from `stdin`, skipping the following actions.

5. Decrease the `key` field of the corresponding object to `<NewKey>` and perform the corresponding operations on the min heap.

6. Wait for the next instruction from `stdin`.

- `Unknown Instructions`:
  If your program reads in an unknown instruction (other than the ones listed in the above), the program should do the following.

  1. Write the following message to `stdout`

     `Warning: Invalid instruction`

  2. Wait for the next instruction from `stdin`.

4 Format of the Input File

The input file specified by `argv[1]` is an ascii file. It contains an integer n, followed by n real numbers, where two numbers are separated by one or more white spaces. Here a white space one of the three characters in the set

`{' ', '\t', '\n'}`.

The first integer, call it n, is the number of key values. The next n real numbers are the first key value, the second key value, ..., the n-th key value. An example input file `input` has the following content.

```
7 17 16
15
14 13 12 11
```

It contains the same information as the following file:

```
7
17.0
16
15
14.00
13
12
11
```

5 Submission

You should submit your project to Gradescope via the link on Canvas. Submit your Makefile along with all header files and implementation files. You should put your name and ASU ID at the top of each of the header files and the implementation files, as a comment.

Submissions are always due before `11:59pm` on the deadline date. Do not expect the clock on your machine to be synchronized with the one on Canvas/Gradescope. This project is due on Wednesday, 10/15/2023. It is your responsibility to submit your project well before the deadline. **Since you have more than 20 days to work on this project, no extension request (too busy, sick, need more time accommodations) is a valid one.**
The instructor and the TAs will offer more help to this project early on, and will not answer emails/questions near the project due date that are clearly in the very early stage of the project. So, please manage your time, and start working on this project immediately. **You are requested to submit a version of your project on Gradescope on each of the Check Point dates.**

6 Grading

All programs will be compiled and graded on Gradescope. **If your program does not compile and work on Gradescope, you will receive** 0 **on this project**. If your program works well on `general.asu.edu`, there should not be much problems. The maximum possible points for this project is 100. The following shows how you can have points deducted.

1. **Non-working program**: If your program does not compile or does not execute on Gradescope, you will receive a 0 on this project. Do not claim "my program works perfectly on my PC, but I do not know how to use Gradescope."

2. **Posted test cases**: For each of the 20 posted test cases that your program fails, 4 points will be marked off.

3. **UN-posted test cases**: For each of the 5 un-posted test cases that your program fails, 4 points will be marked off.

7 Check Points

While you have a lot of time to work on this project, time goes very fast. You are required to submit a version of your project on Gradescope on each of the given Check Point days.

8 Examples

In this section, I provide some examples. All examples assume that the input file is named `FileI` and has the following content:

```
9


1
2
3
8
5
6
7
4
9
```

8.1 Example 1

Execution line is the following:

```
./PJ2
```

This is an invalid execution. The program writes the following error message to `stderr` and terminates.

```
Usage: PJ2 <FileI> <FileO> flag
<FileI>    is the input file
<FileO>    is the input file
flag       is in {0, 1, 2, 3}
```

8.2 Example 2

Execution line is the following:

```
./PJ2 FileI my-FileO 0
```

The instructions from `stdin` are as follows:

```
Init 10
Read
Print
Write
Stop
```

This is a valid execution. The program writes the following to `stdout`:

```
Instruction: Init 10
Instruction: Read
Instruction: Print
9
1.000000
2.000000
3.000000
4.000000
5.000000
6.000000
7.000000
8.000000
9.000000
Instruction: Write
Instruction: Stop
```

writes the following to the file `my-FileO`:

```
9
1.000000
2.000000
3.000000
4.000000
5.000000
6.000000
7.000000
8.000000
9.000000
```

and terminates.

8.3 Example 3

Execution line is the following:

```
./PJ2 FileI my-FileO 1
```

The instructions from `stdin` are as follows:

```
Init 10
Read
Print
Write
Stop
```

This is a valid execution. The program writes the following to `stdout`:

```
Instruction: Init 10
Instruction: Read
Input array is NOT a MinHeap
Call BuildHeap
Number of Heapify calls triggered by BuildHeap: 5
Instruction: Print
9
1.000000
2.000000
3.000000
```

```
4.000000
5.000000
6.000000
7.000000
8.000000
9.000000
Instruction: Write
Instruction: Stop
```

writes the following to the file `my-File0`:

```
9
1.000000
2.000000
3.000000
4.000000
5.000000
6.000000
7.000000
8.000000
9.000000
```

and terminates.

9 Test Cases and Autograder

There are 20 posted test cases and 5 hidden test cases. The autograder is also set up on Gradescope.