CSE310 Project 2: Max Heap, Min Heap, and Double-Ended Heap
Due: 10/24/2022, Posted: 09/30/2022, Updated: 10/13/2022.

This should be your individual work. 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`.

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 addition to the above, you have to implement the following four functions (two for max heap and two for min heap), as specified in the following.

1. For **max heap**, you need to implement `DecreaseKey` and `Delete` as specified in the following.

```
DecreaseKey(A, pos, newKey){
    if ((pos is a valid index) && (A[pos] > newKey)){
        A[pos] = newKey;
        MaxHeapify(A, pos);
    }
}

Delete(A, pos){
    if (pos is a valid index){
        element = A[pos];
        last = heap_size;
        A[pos] = A[heap_size];
        heap_size = heap_size - 1;

        If (pos > 1 && pos <= heap_size && A[pos] > A[pos/2]){
            while (pos > 1 && A[pos] > A[pos/2]){
                exchange A[pos] and A[pos/2];
                pos = pos/2;
            }
        }else{
            MaxHeapify(A, pos);
        }
        return element;
    }
}
```

2. For **min heap**, you need to implement `IncreaseKey` and `Delete` as specified in the following.

```
IncreaseKey(A, pos, newKey){
    if ((pos is a valid index) && (A[pos] < newKey)){
        A[pos] = newKey;
        MinHeapify(A, pos);
    }
}

Delete(A, pos){
    if (pos is a valid index){
        element = A[pos];
        last = heap_size;
        A[pos] = A[heap_size];
        heap_size = heap_size - 1;

        If (pos > 1 && pos <= heap_size && A[pos] < A[pos/2]){
            while (pos > 1 && A[pos] < A[pos/2]){
                exchange A[pos] and A[pos/2];
                pos = pos/2;
            }
        }else{
            MinHeapify(A, pos);
        }
        return element;
    }
}
```

For ease of presentation, both in class and in the above, we presented the functions for **max heap** and **min 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 `int`, along with other fields. Therefore, the heap order is decided by `A[i]->key` or `A[i].key`, depending on your implementation.

1.1 Data Types

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

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

```
typedef struct TAG_HEAP{
    int capacity;  /* capacity of the (max heap)/(min heap)/(double-ended heap ) */
    int size;      /* current size of the heap */
    ELEMENT **A;   /* array of pointers to ELEMENT */
    ELEMENT **a;   /* array of pointers to ELEMENT */
    other fields as you see fit
}HEAP;
```

You can use the data type `HEAP` defined in the above to implement max heap, min heap, and double-ended heap (that supports max heap operations as well as min heap operations). Essentially, the max heap operations operate in the array `A` and the min heap operations operate in the array `a`. The use of the `HEAP` data structure can be controlled by a parameter `ADT`, which can take one of the values in $\{1, 2, 3\}$. When the value of `ADT` is 1, `HEAP` is just like a max heap (without using `A`). When the value of `ADT` is 2, `HEAP` is just like a min heap (without using `a`). When the value of `ADT` is 3, `HEAP` can be used as a double-ended heap. Therefore, you need to include `ADT` as a parameter in all of the heap related functions you implement. **The value of `ADT` will stay the same throughout the execution of your program**. In particular,

- When `ADT` is 1, we will only perform max heap operations.
- When `ADT` is 2, we will only perform min heap operations.
- When `ADT` is 3, we will perform both max heap operations and min heap operations.

2 Modular Design

You will continue to have 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 max heap, min heap, and double-ended 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.

3 Flow of the Project

3.1 Valid Execution

A valid execution of your project has the following form:

`./PJ2 DataStructure Capacity`

where `./PJ2` tells the system to search for the executable file `PJ2` in the current directory, The value of `DataStructure` should be in the set {`MaxHeap`, `MinHeap`, `DoubleHeap`}, indicating which data structure to use, and `Capacity` is a positive integer, indicating the capacity of the corresponding heap. Your program should check whether the execution is valid. If the execution is not valid, your program should print out the message to `stderr` and stop.

```
Usage: ./PJ2 DataStructure Capacity
       DataStructure should be in {MaxHeap, MinHeap, DoubleHeap}
       Capacity must be a positive integer
```

Note that your program should not crash when the execution is not valid. For a valid execution, your program should set the value of `ADT` to 1 (if `DataStructure` is `MaxHeap`), 2 (if `DataStructure` is `MinHeap`), 3 (if `DataStructure` is `DoubleHeap`), respectively.

Upon a valid execution, your program should initialize the heap (including dynamic memory allocation), then expect the following commands from `stdin` and act accordingly:

- `Stop`:
  On reading `Stop`, the program stops.
- `Print`:
  On reading `Print`, the program should do the following.
  1. Print the current state of the heap to `stdout` (refer to posted test cases for output format).
  2. Wait for the next command from `stdin`.
- `Read`:
  On reading `Read`, the program should do the following.
  1. Open the file "HEAPifile.txt" in read mode.

     If the file is not opened successfully, write an error message to `stderr` and wait for the next command from `stdin`, skipping the following actions.
  2. Read in the first integer, `n`, from the file opened.

     If (`heap` is `NULL` or `heap->capacity` is smaller than `n`), write an error message to `stderr`, close the file `HEAPifile.txt`, and wait for the next command from `stdin`, skipping the following actions.
  3. For each value of $j = 1, 2, \ldots, n$, read in the integer key_j from the 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/arrays in the heap point to this object, depending on the value of `ADT`. Close the file `HEAPifile.txt`.

4. Apply the $O(n)$ time `BuildHeap` function to build the max heap or min heap or double-ended heap, depending on the value of `ADT`.
5. Wait for the next command from `stdin`.

- `Write`:
  On reading `Write`, the program should do the following.
  1. Opens the file "HEAPofile.txt" in write mode.

     If the file is not opened successfully, write an error message to `stderr` and wait for the next command from `stdin`, skipping the following actions.
  2. Write the heap information to the file "HEAPofile.txt" (refer to posted test cases for format). Close the file `HEAPofile.txt`.
  3. Wait for the next command from `stdin`.

- `Insert Key`:
  On reading `Insert Key`, where `Key` is an integer, the program should do the following.
  1. If the heap is full, write an error message to `stderr` and wait for the next command from `stdin`, skipping the following actions.
  2. Dynamically allocate memory for an object of type `ELEMENT`. Set the `key` field of this object to `Key`. Insert this object to the heap, according to the value of `ADT`.
  3. Wait for the next command from `stdin`.

- `ExtractMax`:
  On reading `ExtractMax`, the program should do the following.
  1. If (the value of `ADT` is 2 `or` the heap is empty), write an error message to `stderr` and wait for the next command from `stdin`, skipping the following actions.
  2. Perform the ExtractMax operation (depending on whether the value of `ADT` is 1 or 3). Write the key value of the extracted object to `stdout` using the format
     `fprintf("ExtractMax: %d\n", Key);`
     where `Key` is the value of the `key` field of the extracted object.
  3. Wait for the next command from `stdin`.

- `ExtractMin`:
  On reading `ExtractMin`, the program should do the following.
  1. If (the value of `ADT` is 1 `or` the heap is empty), write an error message to `stderr` and wait for the next command from `stdin`, skipping the following actions.
  2. Perform the ExtractMin operation (depending on whether the value of `ADT` is 2 or 3). Write the key value of the extracted object to `stdout` using the format
     `fprintf("ExtractMin: %d\n", Key);`
     where `Key` is the value of the `key` field of the extracted object.

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

- `IncreaseKey Postion NewKey`:
  On reading `IncreaseKey Position NewKey`, the program should do the following. Here `Position` is the index to the array for the **max 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. If (the value of `ADT` is 2 `or` the heap is empty `or Position` is an invalid index to the heap array `or NewKey` is not larger than the `key` field of the corresponding object), write an error message to `stderr` and wait for the next command from `stdin`, skipping the following actions.
  2. Increase the `key` field of the corresponding object to `NewKey` and perform the corresponding operations on the max heap (if the value of `ADT` is 1) or the double-ended heap (if the value of `ADT` is 3).
  3. Wait for the next command 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. If (the value of `ADT` is 1 `or` the heap is empty `or Position` is an invalid index to the heap array `or NewKey` is not smaller than the `key` field of the corresponding object), write an error message to `stderr` and wait for the next command from `stdin`, skipping the following actions.
  2. Decrease the `key` field of the corresponding object to `NewKey` and perform the corresponding operations on the max heap (if the value of `ADT` is 1) or the double-ended heap (if the value of `ADT` is 3).
  3. Wait for the next command from `stdin`.

4 Format of the Input File

The file `HEAPifile.txt` is an ascii file. It contains many integers, where two integers 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 integers are the first key value, the second key value, ..., the n-th key value. An example for HEAPifile.txt is

```
5   1 2 3
4
5
```

It contains the same information as the following file:

```
5
1
2
3
4
5
```

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 Monday, 10/24/2022. It is your responsibility to submit your project well before the deadline. **Since you have more than three weeks to work on this project, no extension request (too busy, other business, sick, need more 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 today**.

Please save a copy of your project on `general.asu.edu`. In rare cases, we may be willing to take a look at the time stamps of your files on `general.asu.edu` as well as the content of your project. We will not check these on any other machine, not your own computer for sure.

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. **Program requires non-standard libraries**: If your program does not compile/working using the compiler and flags as stated in the sample makefile, 20 points will be marked off.

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

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

7 Test Cases

Posted test cases will be posted on Canvas shortly.

8 Suggested Steps

Some students may not have a lot of experience writing codes that can read commands from `stdin`. It is suggested that you start your project by writing a framework that can read in the list of valid commands (without taking the corresponding actions). After this is done, you can add on one piece at a time.

If you cannot successfully implement the double-ended heap, you can implement a max heap and a min heap. In this case, you will at least earn the credit for the cases where the value of `ADT` is 1 or 2.

9 Extra Help

The following piece of code may be helpful to some of you.

```
#include <stdio.h>
#include <string.h>
int main(){
    char word[50];
    int value;
    while (1){
        scanf("%s", word);
        if (strcmp(word, "Insert")==0){
            scanf("%d", &value);
            printf("Command: %s %d\n", word, value);
        }else{
            printf("Command: %s\n", word);
        }
    }
    return 0;
}
```

10 Output Format

10.1 Output Format for `Print`

If the heap is empty (size=0), the program should print one line to `stdout` using the format:

```
printf("capacity=%d, size=%d\n", heap->capacity, heap->size);
```

If the heap is not empty, and `ADT`=1, the program should print two lines to `stdout`, where the first line is as in the above, and the second line begins with the string `MaxHeap:` which is followed by the key values of the objects pointed to by the i-th node in the max heap, $i = 1, 2, \ldots,$ `heap->size`, with a comma followed by exactly one space in between.

If the heap is not empty, and `ADT`=2, the program should print two lines to `stdout`, where the first line is as in the above, and the second line begins with the string `MinHeap:` which is followed by the key values of the objects pointed to by the i-th node in the min heap, $i = 1, 2, \ldots,$ `heap->size`, with a comma followed by exactly one space in between.

If the heap is not empty, and `ADT`=3, the program should print three lines to `stdout`, where the first line is as in the above, the second line begins with the string `MaxHeap:` which is followed by the key values of the objects pointed to by the i-th node in the max heap, $i = 1, 2, \ldots,$ `heap->size`, with a comma followed by exactly one space in between; the third line begins with the string `MinHeap:` which is followed by the key values of the objects pointed to by the i-th node in the min heap, $i = 1, 2, \ldots,$ `heap->size`, with a comma followed by exactly one space in between.

10.2 Output Format for `Write`

The format for `Write` is identical to that for `Print`. The only difference is that `Write` writes to a the file `HEAPofile.txt` while `Print` writes to `stdout`.

10.3 Error Messages

Error messages should be written to `stderr`, rather than `stdout`. Besides the ones explained earlier, the following are the formats for possible error messages.

```
fprintf(stderr, "Error: Cannot allocate memory for heap\n");
fprintf(stderr, "Error: Cannot allocate memory for max heap array\n");
fprintf(stderr, "Error: Cannot allocate memory for min heap array\n");
fprintf(stderr, "Error: Cannot open file HEAPifile.txt\n");
fprintf(stderr, "Error: Cannot open file HEAPofile.txt\n");
fprintf(stderr, "Error: DecreaseKey in a max heap or a null/empty heap\n");
fprintf(stderr, "Error: ExtractMax in a min heap or a null/empty heap\n");
fprintf(stderr, "Error: ExtractMin in a max heap or a null/empty heap\n");
fprintf(stderr, "Error: Heap overflow\n");
fprintf(stderr, "Error: IncreaseKey in a min heap or a null/empty heap\n");
fprintf(stderr, "Error: Insertion to a null/full heap\n");
fprintf(stderr, "Error: Invalid command\n");
fprintf(stderr, "Error: Invalid position or newKey in DecreaseKey\n");
fprintf(stderr, "Error: Invalid position or newKey in IncreaseKey\n");
```

These are self-explanatory.

11 Understanding the Test Cases

The *posted test cases* have been posted on Canvas, as a zip file. Each of the test cases has its own directory and have certain (ascii) files in its own directory. These files include the following:

```
Commands
Execution
HEAPifile.txt
HEAPofile.txt
Message.txt
Output.txt
```

We use test case 01 to explain the meaning of these files.
The file `Commands` has the following content:

```
Print
Read
Print
Write
Stop
```

The file `Execution` has the following content:

```
#!/bin/bash
./PJ2 DoubleHeap 80 < Commands > Output.txt 2> Message.txt
```

This shell script has four parts:

- `./PJ2 DoubleHeap 80`
- `< Commands`
- `> Output.txt`
- `2> Message.txt`

where the first part is a valid execution of the program; the second part redirects the content of file `Commands` to `stdin`; the third part redirects `stdout` to file `Output.txt`; and the fourth part redirects `stderr` to file `Message.txt`.

The effect is similar to running the program using the command

```
./PJ2 DoubleHeap 80
```

then typing each line in the file `Commands` through the keyboard.

The file `HEAPifile.txt` has the following content:

```
6

1
7
6
3
5
4
```

The file `Output.txt` has the following content:

```
capacity=80, size=0
capacity=80, size=6
MaxHeap: 7, 5, 6, 3, 1, 4
MinHeap: 1, 3, 4, 7, 5, 6
```

The files `HEAPofile.txt` and `Message.txt` are also in the directory.