CSE310 Assignment #3

Due Date/Time: Friday, Oct. 2, 2020 by 11:59PM

General Requirements for Assignment

- *All assignments in this course must be individual work*, *i.e.* you are required to compose your own unique solution to each assignment.
- We will use powerful plagiarism detection software to compare and check all students submission, cheating will be reported to the Dean's office!
- It must be submitted online through gradescope! *We don't accept submissions through emails*.
- Submission link will be closed automatically once the due date/time is past. *Strictly no late assignment will be accepted*.
- For each assignment, you will have unlimited times to submit before the due date/time, but we will only check and grade your last submission.

1. Objectives

- Review on dynamic memory allocation in C++ for 1D array
- Implement the max heap data structure.
- Implement heap operations such as insert, extracting the root, search, increase key, max-heapify etc.
- Review on using basic Linux commands to compile, run and check your program at *general.asu.edu*.

2. Given file(s) in C++

- Assignment3.cpp (Driver program, partiall given, need to be modified)
- Heap.h (Header file. More code need to be added)

Study the following sample code to learn how to dynamically allocate memory for an array of objects, etc.

- DynamicArrayDemo.cpp

3. Assignment Description

Write a C++ program that creates a **Heap** data structure that contains various **Food**. **Food** is a structure defined as below:

```
struct Food
{
    int key;
    string foodName;
    double price;
};
```

The ***key*** of a food will uniquely identify that food, *i.e.* no two Food objects contain the same key.

The **Heap** class defined inside the Heap.h file is a class that contains the following three (**3**) instance variables, namely *foodArr*, *capacity*, *size* and various functions defined below.

```
struct Food* foodArr;
int capacity;
int size;
```

foodArr is a one dimensional array containing **Food** objects. It will be created with the *capacity* by using dynamic memory allocation. The *capacity* of the **Heap** represents the maximum number of Food objects that can be stored inside the Heap; the *size* of the **Heap** represents the actual number of Food objects stored inside the Heap. When a heap size reaches its capacity (*i.e. size* equals *capacity*), the heap should automatically double its original *capacity* and allows more Food objects to be inserted inside, also during this procedure, the original heap contents and structure should be maintenained.

You will need to create the following functions in the **Heap** class (within Heap.h file). ***I highly recommend that you read textbook pp.151 ~ 164 to study the psedocode or algorithms related with those heap operations,*** they will help you developing the actual code. Also follow the instructions closely, otherwise you might not be able to pass certain special test cases!

Function	Function's Description
`Heap(int capacity)`	This is the constructor, it will initialize *foodArr* and dynamically allocate memory that be able to hold *capacity* number of Food objects in the heap. It should also initizlize the *size* of the heap to be 0.
`~Heap()`	This is the destructor, it should delete the memory allocate for the heap and print the following message on screen. `The number of deleted food items is: ??` ?? should be the actual number of heap nodes being deleted.
`Food* getFoodArr()`	This is the accessor for the instance variable *foodArr*.
`int getSize()`	This is the accessor for the instance variable *size*.
`int getCapacity()`	This is the accessor for the instance variable *capacity*.
`int isFound(int foodKey)`	Given a key, this method returns the Food object's index inside the *foodArr* if it find a food with the specific *foodKey*; otherwise it should return -1.
`bool increaseKey(int index, Food foodwithNewKey)`	Given an *index* and a Food object called *foodwithNewKey*, this function **increases** the Food object at *index*'s key to the new key as inside the *foodWithNewKey.* The function returns *true* if the operation is successful and *false* otherwise. **Note**: after the key increase operation, the max heap properties need to be hold, *i.e.* the newly increased key object might need to be floated up in order to keep the max-heap properties.
`bool insert(int key, string foodName, double foodPrice)`	Given the information of a Food object, namely its *key*, *name* and *price*, this function create a new Food ojbect and inserts it inside the heap. If successful, it should update the *size* and return *true*; otherwise return *false*.

	Note: according to max heap operation, when we insert a new node, we should always insert it to be the right-most node of the last level, then if necessary, we need to "float" it up to keep the max-heap properties. You can first set an extremly "small" dummy Food object to be the last object inside the heap, *i.e.* at index *size*-1, then call above *increaseKey*( ) function to increase the dummy object's key to be the new key (read textbook pp. 164 for the algorithm).
	
	In case the heap's size reaches its capacity, *i.e. size* equals *capacity*, you should dynamically re-allocate memory for the heap and **double** its original capacity, then insert the new Food object inside. Inside this function, it should show the following message on screen. During the procedure, the original heap elements and properties should be kept.
	
	`Reach the capacity limit. Double the capacity`
	`The new capacity now is ??`
	
	where ?? should be the new doubled capacity.

`void heapify(int index)`	As we learned in class, before we call this *heapify*( ) function, we assume that the binary tree rooted at left and right child of the node at *index* are already max heaps, but node at *index* might be samller than its children, thus violating the max-heap property. *heapify*( ) lets the node at *index* "float down" in the max-heap so that the subtree rooted at *index* obeys the max-hea property.
`Food getHeapMax()`	This is the accessor for the root of the heap, it returns the largest Food object (also root) of the max-heap. Note: this function only get the root node information of the max-heap, it does NOT remove it.
`void extractHeapMax()`	This function extracts the root of the heap. Basically we replace the root by the last node of the heap, then call *heapify*( ) to "float-down" the newly added root, we then decrease the *size* of the heap by 1.
`int leftChild(int parentIndex)`	Given a parent node's index, this function returns its left child's index inside the 1D array.
`int rightChild(int parentIndex)`	Given a parent node's index, this function returns its right child's index inside the 1D array.
`int parent(int childIndex)`	Given a node's index, this function returns its parent node's index inside the 1D array.

`void printHeap()`	This function uses the breadth-frist traversal to print out the contents of the heap. It should print:
	
	`Heap capacity = ??`
	`Heap size = ??`

	It then print each Food object with the following format (check the solution output for sample output)
	
	`cout << left;`
	`cout << setw(5) << oneFood.key`
	`<< setw(8) << oneFood.foodName`
	`<< setw(8) << fixed <<`
	`setprecision(2) << oneFood.price <<`
	`endl;`

You are given a partially finished driver prgroam called **Assignment3.cpp**, which has the main() method to create a new Heap object and to test those functions defined in Heap class (see sample input & output files). //---- is where you need to fill in your own codes according to the instructions. The Assignment3.cpp will display the following menu to the user:

```
Choice Action
------ ------
C Create a heap
D Delete the heap
E Extract max node
F Find a food by key
I Insert a food
K Increase the key
M Get the max node
P Print the heap
Q Quit
? Display Help
```

The program will ask “What action would you like to perform? ”. A user will type in a character of their menu choice. Note: user might enter both upper case or lower case letters. Please find below for each command's description

Command	**Command's Description**
'**C**'	This command will ask user to enter the capacity of a heap, it then create and intialize an **Heap** object with the specific *capacity*.
'**D**'	This command call the destructor explicitly to delete the existing/current heap and re-initialize and create one new **Heap** object with the initial capacity set to be the default **5**. This command allows user to delete and create multiple heaps within a single test case.

'**E**'	This command extracts the root (also the max node) from the heap. If the heap is empty or hasn't been initialized yet, it should show the following message on screen:
	
	`Empty heap, can NOT extract max`
	
	If the heap is non-empty, first it will print:
	
	`Before extract heap max operation:`

	followed by the contents of the heap **before** the extracting root operation. It then should call the extracting root operation, last it will print the message: `After extract heap max operation:` followed by the contents of the heap **after** the extracting root operation so that a user can verify that the root is indeed removed successfully.

'**F**'	This command ask user to enter a Food object's key, it then search the heap by using the key, if find the relevant Food object with the relevant key inside the heap, it print the following message on screen:
	
	`Food with key: ?? is found`
	
	Otherwise it print:
	
	`Food with key: ?? is NOT found`
	
	`??` is the key user entered for searching.

'**I**'	This command inserts a new Food object inside the heap. It asks the user to enter the food's name, key and price, it then insert the relevant food inside the heap. If insertion is successful, it prints, for example:
	
	`The food "bacon" is added`
	
	Otherwise, print for example:
	
	`The food "bacon" is NOT added`
	
	where `"bacon"` can be any other food's name we might want insert inside the heap.

'**K**'	This command increases a Food object's key. The command will first ask the user to enter the old food's key to increase, it then ask the user to enter the new key. In case the new key is less than the old key, the function should show the following message on screen.
	
	`Increase key error: new key is smaller than current key`
	
	In case the Food object with the old key doesn't exist inside the heap, the command should print:
	
	`The old key you try to increase does not exist`
	
	Or in case the new key the user entered already exist inside the heap, the command should print:
	
	`The new key you entered already exist, increase key operation failed`
	
	Otherwise first print the following message on screen:
	
	`Before increase key operation:`
	
	Followed by calling *printHeap*() to print out the contents of the heap **before** the increasing key

	operation.
	
	It then print the following message on screen:
	
	`Food with old key: ?? is increased to new key: ??`
	
	`After increase key operation:`
	
	Followed by calling *printHeap*() again to print out the contents of the heap **after** the increasing key operation so that you can verify the operation.

'M'	This command allow us to get the the max node's info. of the heap (also the root). It should print out the Food object, such as the following one:
	
	`The maximum heap node is:`
	`25  chips  2.99`
	
	The format for the maximum node is:
	
	`cout << setw(5) << maxFood.key`
	`  << setw(8) << maxFood.foodName`
	`  << setw(8) << fixed << setprecision(2) << maxFood.price << endl;`
	
	If the heap is empty, the command should print:
	
	`Empty heap, cannot get max node`

'P'	The command call the *printHeap*() function on the heap and prints out the heap contents. If the heap is empty, it should show the following message on screen: `Empty heap, no elements`
'Q'	This command deletes the heap object we created and terminate the driver program.
'?'	This command prints out the menu.

4. Sample Run

Click here to see a sample run of the program.

Note: if there's any discrepency betweent above sample run and the test case (it may happen due to typo), please use the test cases output as the final guide since they are the one used to check against your submission.

5. Other Requirements

- You need to implement this program by using C++ and your program need to compile, execute and pass the test cases on ***general.asu.edu ,*** in a Linux/Unix environment before you submit it on gradescope.com. So start working today, and do not claim that "*my program works perfectly on my PC, but I do not know how to use general.asu.edu*" or "*my program works perfectly on my PC, but when I test it on general.asu.edu, it gave me segmentation error*", etc.

- Mac users, your operating system already has a SSH client installed, use it directly, Mac OS is a kind of Linux/Unix based, so compile/run your C++ code by openning a terminal window directly.
- Windows 10 users, if you already have or used a SSH client (such as PuTTY), use it freely, otherwise consider installing WinSCP, it's a file transfer client bundled with PuTTY. Or if you want to compile and run your code directly on your own PC, Windows 10 comes with an option that allows you to run Linux distro, such as Ubuntu on it, check the following website to see instrucitons on how to set it up:

  https://www.youtube.com/watch?v=X-DHaQLrBi8

- For this assignment, ***you are NOT allowed to use any predefined data structures*** (such as vector, list in C++ STL etc.) except arrays and strings, you need to build your own heap data structures and operations associated with it (such as insertion, extraction, searching, etc).Your code should be well documented and commented. The assignment number, your name, StudentID, and a class description need to be included at the top of each class/file.

```
// Assignment : Arizona State U. CSE310 #3
// Name : Your name
// StudentID : Your ID
// Email : Your email address
// Description :
```

6. Test and Run Your Program

Using the following four test cases to test and check your program at ***general.asu.edu***, to compile your program, you could type, for example:

```
xfeng13@general2:~/cse310/Assignment/HW3$ g++ -std=c++11 -o prog
Assignment3.cpp Heap.h
```

To run, type:

```
xfeng13@general2:~/cse310/Assignment/HW3$ ./prog <input1.txt >myout1.txt
```

Then compare your program's ouput, namely *myout1.txt* with our solution output file *output1.txt* by using a file comparison program, such as diffMerge or the following online one. To pass the four test cases on gradescope, your program's output need to be the same as our solution output. To make the matching task a little easier, when checking your program's output against our solution output, we will ignore all empty spaces and only check your program's output computation results (characters). Make sure they are identical.

https://www.quickdiff.com/

Input

The following files are the test cases that will be used as input for your program (Right-click and use "Save As"):

Test Case #1
Test Case #2
Test Case #3
Test Case #4

Output

The following files are the expected outputs of the corresponding input files from the previous section (Right-click and use "Save As"):

Test Case #1
Test Case #2
Test Case #3
Test Case #4

7. Submission

1) Test your program by using above four test cases, compare your program's output with our solution output, make sure they contain exactly the same strings/characters before you submit the source file **Assignment3.cpp** and **Heap.h**. Note: even one signle character difference will cause your program fail all test cases on submission server!

2) To submit, login to our Canvas course website, from left hand side control panel, click on gradescope, it will bring you to a seperate gradescope.com website. Submit ONLY the source code file(s), namely **Assignment3.cpp** and **Heap.h**, make sure your file's name is EXACTLY **Assignment3.cpp** and **Heap.h** (NOT the assignment3.cpp, HW3.cpp or Heap.hpp) , nothing else! You need **NOT** to submit the test cases.

3) Once inside gradescope.com, from left hand side, click on Assignments, you will notice that HW3 is already posted inside, click on HW3."Click to browse" to find where you saved the Assignment3.cpp and Heap.h file.Upload the files to the site and then click on the green "Upload" button located at the bottom to submit.If you submit the correct file(s) and your program compiles successfully and generate the expected output, you will see a green message with 16 pts assigned automatically; otherwise you will see red color message, it indicates that your program failed one or more test cases, in that case you will need to make any necessary changes to the original source code and click on the low right corner "Resubmit" button to re-submit your modified source code(s).

4) For all assignments, you have unlimited times to resubmit before the due date and time, **but we will only check and grade your last submission**! To pass the four auto test cases on gradescope, your program's output need to be the same as our solution output. If you can't get your program to pass the test cases, submit whatever you've done so far and we do give partial credits!

5) It's your responsibility to make sure that you submitted the correct file on server **BEFORE** the due date and time. After the deadline, *we will **NOT** accept any submissions through emails, No exception!*

8. Grading Rubric (Total 30 pts)

- ____/ 1 Documentation (header including a description and comments for each function, class and file)
- ____/ 1 Indentation and Spacing (easy to read)
- ____/ 1 the destructor `~Heap()` function is defined correctly (and it produces the correct output)
- ____/ 1 the `isFound` function is defined correctly (and it produces the correct output)
- ____/ 2 the `increaseKey` function is defined correctly (and it produces the correct output)
- ____/ 2 the `insert` function is defined correctly (and it produces the correct output)
- ____/ 2 the `heapify` function is defined correctly (and it produces the correct output)
- ____/ 2 the `extractHeapMax` function is defined correctly (and it produces the correct output)
- ____/ 2 the `printHeap` function is defined correctly (and it produces the correct output)
- ____/ 16 Correct output for the four test cases