Asgn 4 - CS3310 Spring 2016
Dr. Kaminski

using a **Priority Queue**
implemented as a **Max Heap**
[an **Internal Data Structure**]

OVERVIEW: This project is a **single** batch processing program with a separate OOP *class (in a separate code file)* for the PriorityQueue (PQ) storage and handling. The main program *(including its local private methods)* is the overall controller which reads and manages getting the requested tasks processed. The 5 types of tasks (specified in the TaskList file) include:

1. **build** the PQ - using designated data from the InputStream file
2. **add** items to the PQ - using designated data from the InputStream file
3. **remove** items from the PQ – the number specified
4. **empty** the PQ completely
5. **snapshot -** display PHYSICAL implementation of the PQ (array view, not heap view) – a utility module for the developer.

********** THE 3 DATA FILES **********

INPUT: `TaskList.csv` contains a sequence of tasks to be done in the order specified. These requests are all one of the 5 types of tasks described below. This file guides the main controller's execution of the whole program. A big switch statement based on the TransCode controls which method in the PriorityQueue class is called.
SEE PROCESSING NOTE BELOW

INPUT: `InputStream.csv` contains the data to be stored in the PQ (in response to build and add requests, guided by their specified parameters for build requests).

- Each record contains name, continent, region, population (in that order).
- Only name & population are actually stored in the PQ
- Population is used as the priority value (PV).
- Continent & region are just used for control purposes.

SEE PROCESSING NOTE BELOW
NOTE: Data records in this file are in the order needed for the proper handling of the specific order of the task requests in the given TaskList file.

OUTPUT: `Log.txt` 1) Task Request is echoed, 2) then appropriate response is shown.

PROCESSING NOTE FOR BOTH INPUT FILES: The files are STREAMS of data, not data STORES, per se. So the program reads through the file ONCE, using the StreamProcessing algorithm ("design pattern") for processing a stream of data. – that is,
Loop { read 1 record, completely deal with it }
It is NOT all read into memory & stored in a big array, to be processing from the array as input.

********** 5 Types of TASKS **********

NOTE: Data shown below isn/t accurate for the given input – it's just shows required formatting
NOTE: For REMOVE and EMPTY and SNAPSHOT,
1 - don't show the . . . part as I did below – actually show all the required countries
2 - Longest name is 28 char's & biggest population is 9 digits.
Right-justify populations and include commas

`BUILD,continent,`*nameOfContinent*

- read & store countries in the PQ which match the specified nameOfContinent UNTIL there's a "break" - i.e., when a non-matching continent is encountered, temporarily suspend inputting the stream
- CAUTION: Don't lose the country at the "continent-break" when the program resumes processing – i.e., the country that's already been read in, but not yet used, because it's a different continent - and so the 1st country to use when processing resumes has ALREADY been read into memory!!!
- USES: `pq.add` method (called repeatedly, 1 call per country to be inserted)
- TASK's RESPONSE to Log (including echo of request):

```
BUILD,continent,South America
     >>>> 14 countries STORED in PQ
```

`ADD,region,`*nameOfRegion*

- read & store countries in the PQ which match the specified nameOfRegion until there's a "break" - i.e., when a non-matching region is encountered, temporarily suspend inputting the stream
- see CAUTION above (for build)
- USES: `pq.add` (called repeatedly, 1 call per country to be inserted)
- TASK's RESPONSE to Log (including echo of request)::

```
ADD,region,Western Europe
     >>>> 12 countries ADDED to PQ
```

`REMOVE,`*number*

- delete designated number (M) of countries from PQ - i.e., the N biggest ones currently in the PQ (which'll appear in decreasing population order)
- USES: `pq.remove` (called repeatedly, N times)
- TASKS's RESPONSE to Log (including echo of request)::

```
REMOVE,9
     >>>> 9 countries REMOVED from PQ (in this order)
     01 > United States                > 278,357,000
     02 > Russian Federation           > 146,934,000
     . . .
```

EMPTY

- delete ALL countries currently in the PQ (which'll appear in decreasing population order)
- USES: `pq.empty` (called only once – the method does its own looping)
- TASK's RESPONSE to Log (including echo of request)::

EMPTY

```
>>>> 28 countries REMOVED from PQ (in this order)
01 > United States                > 278,357,000
. . .
27 > Falkland Islands             >       2,000
28 > Vatican City                 >       1,000
```

SNAPSHOT

- show physical array storage to aid developer
- Note: these will NOT be in sorted order – it's a HEAP
- USES: `pq.snapshot` (only once – the method does its own looping)
- TASK's RESPONSE to Log (including echo of request)::

SNAPSHOT

```
N is 14
[00] United States                > 278,357,000
. . .
[13] Belgium                      >  10,239,000
```

********** **THE PRIORITY QUEUE CLASS** **********

- All storage/handling of the PQ / heap MUST be done in the PriorityQueue OOP class.
- The PQ MUST be implemented as a Max Heap (which is a binary tree (BT)).
- The heap MUST use a Linear implimentation of a BT (with implicit ptrs)
  (Do NOT use a Linked implementation, with explicit Ptrs – else 0 points).
  (Do NOT use a Linear implementation of the PQ – else 0 points).
- Linear: array of heapNodes (or 2 parallel arrays) containing:
  name & priorityValue (which is population).
- A heap also needs N, but does not need RootPtr since that's always 0.
- public service methods:
  - constructor – uses heapInitialize
  - add – uses heapInsert (which uses walkup)
  - remove – uses heapDelete (which uses walkdown)
  - empty – repeatedly calls heapDelete
  - snapshot – uses a for loop (using N, not MAX_N) to show array(s)
- private methods:
  - heapInsert & heapDelete
  - walkUp & walkDown

********** **SOME NOTES** **********

- A method should not be longer than a page/screen (ish). If any are longer, modularize further.
- The main program knows that a PriorityQueue is being used, but it does NOT KNOW that the PQ was implemented as a heap. So in terms of public/private designations and the naming of PUBLIC methods in the PriorityQueue class, do NOT mention anything to do with heaps – like heapInsert, heapDelete, walkUp, walkDown, etc. – those would be PRIVATE methods in the PriorityQueue class.
- Method names should match what the design specs use to describe functions, so everyone's "on the same page" and can more easily communicate about the app.