CSE310 Programming Project 2
OUT: WED, 10/09/2013
DUE: WED, 10/23/2013

In this programming project, you will implement the data structure max-heap. You have to use the programming language C++, not any other programming language. Also, your program should be based on the g++ compiler on general.asu.edu. **All programs will be compiled and tested on** general.asu.edu. If you program does not work on that machine, you will receive no credit for this assignment. You need to submit your project via the dropbox, using a single file with the format `CSE310-P02-LastName-FirstName.zip`. We will use your electronic submission to compile and run on test cases.

You need to define the following data types.

- ELEMENT is a data type that contains a field named key, which is of type int. In later assignments, you will have to add on other fields to ELEMENT, without having to change the functions. Note that ELEMENT should not be of type int.
- HEAP is a data type that contains three fields named capacity (of type int), size (of type int), and H (an array of type ELEMENT with index ranging from 0 to capacity).

The functions that you are required to implement are

- Initialize(n) which returns an object of type HEAP with capacity n and size 0.
- BuildHeap(heap, A), where heap is a HEAP object and A is an array of type ELEMENT. This function copies the elements in A into `heap->H` and uses the linear time build heap algorithm to obtain a heap of size size(A).
- Insert(heap, k) which inserts an element with key equal to k into the max-heap heap.
- DeleteMax(heap) which deletes the element with maximum key and returns it to the caller.
- IncreaseKey(heap, element, value) which increases the key field of element to value, which should not be smaller than the current value. Note that you have to make necessary adjustment to make sure that heap order is maintained.
- printHeap(heap) which prints out the heap information, including capacity, size, and the key fields of the elements in the array with index going from 1 to size.

You should implement a main function which takes the following commands from the key-board:

- **S**
- **C n**
- **R**
- **W**
- **I k**
- **D**
- **K i v**

On reading **S**, the program stops.
On reading **C n**, the program creates an empty heap with capacity equal to **n**, and waits for the next command.
On reading **R**, the program reads in the array A from file HEAPinput.txt, calls the linear time build heap algorithm to build the heap based on A, and waits for the next command.
On reading **W**, the program writes the current heap information to the screen, and waits for the next command. The output should be in the same format as in the file `HEAPinput.txt`, proceeded by the heap capacity.
On reading **I k**, the program inserts an element with key equal to **k** into the current heap, and waits for the next command.
On reading **D**, the program deletes the maximum element from the heap and prints the key field of the deleted element on the screen, it waits for the next command.
On reading **K i v**, the program increases the key of element with index i to v.

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 contains n integers, one integer per line. These integers are the key values of the n array elements, from the first element to the nth element.

Grading policies: (Sample test cases will be posted soon.)

(10 pts) Documentation: You should provide sufficient comment about the variables and algorithms. You also need to provide a README file describing which language you are using.

(05 pts) Data types: You should define the required data types.

(05 pts) Makefile: Your program should be in at least three modules, and should provide a working makefile. The executable file should be named run.

(05 pts) Initialize

(05 pts) BuildHeap

(05 pts) Insert

(05 pts) DeleteMax

(05 pts) IncreaseKey

(05 pts) printHeap

Above all, you need to write a working program to correctly parse the commands specified in the project. Without this, your program will not be graded.

As an aid, the following is a partial program for reading in the commands from the keyboard. You need to understand it and to expand it.

```
#include "util.h"
//=============================================================================
int nextCommand(int *i, int *v)
{
  char c;
  while(1){
    scanf("%c", &c);
    if (c == ' ' || c == '\t' || c == '\n'){
        continue;
    }
    if (c == 'S' || c == 'R' || c == 'W' || c == 'D'){
        break;
    }
    if (c == 's' || c == 'r' || c == 'w' || c == 'd'){
        break;
    }
    if (c == 'K' || c == 'k'){
        scanf("%d", i); scanf("%d", v);
        break;
```

```
    }
    printf("Invalid Command\n");
  }
  return c;
}
//============================================================================
```

The following is a partial program that calls the above program.

```
//============================================================================
#include <stdio.h>
#include <stdlib.h>
#include "util.h"

int main()
{
    // variables for the parser...
    char c;
    int i, v;
    while(1){
        c = nextCommand(&i, &v);
        switch (c) {
            case 's':
            case 'S': printf("COMMAND: %c.\n", c); exit(0);

            case 'k':
            case 'K': printf("COMMAND: %c %d %d.\n", c, i, v); break;

            default: break;
        }
    }
    exit(0);
}
//============================================================================
```

The following is a partial Makefile.

```
EXEC = run
CC = g++
CFLAGS = -c -Wall

# $(EXEC) has the value of shell variable EXEC, which is run.
# run depends on the files main.o util.o heap.o
$(EXEC) :main.o util.o heap.o
# run is created by the command g++ -o run main.o util.o
# note that the TAB before $(CC) is REQUIRED...
        $(CC) -o $(EXEC) main.o util.o heap.o

# main.o depends on the files main.h main.c
main.o:main.h main.cpp
# main.o is created by the command g++ -c -Wall main.cpp
# note that the TAB before $(CC) is REQUIRED...
        $(CC) $(CFLAGS) main.cpp

util.o  :util.h util.cpp
        $(CC) $(CFLAGS) util.cpp

heap.o  :heap.h heap.cpp
        $(CC) $(CFLAGS) heap.cpp

clean   :
        rm *.o
```