```
Function Pointers, Sorting
Updated 11:30 am 19/March/2018:
 polarity wrt to < > lt gt max min up down

Updated 3:50pm 17/March/2018:
 corrected store/retrieve args

Submit this during your lab session (March 19/20)
with an arg of 7.

Preliminaries
* Read fp.c
* Compile fp.c
* Run fp.c
* What address does a.out indicate for sq and cu?
* run objdump -S a.out
* What address do you see sq and cu at?
* What's the relationship?

* Read Ch 4 of Skiena

A. Parametrized Bubble Sort
Recall how bubble sort works (prior labs, prior solutions,
prior tests, textbook, etc.)

Within the inner for loop of the bubble sort one does a
comparison: based on the type of the comparison, the resulting
array will be sorted in ascending or reverse-ascending order.

One can abstract this even further: instead of just a comparison
function (<, >, etc.) one could write a more complex binary
(that is, two-operand) function that tests for some relationship
between the current element of an array and the succeeding element
of an array. For example, if x[i] < x[i+1]+some_constant, then
swap, etc.

Hence, one can write a parametrized bubble sort algo that will
take as arguments:
  * the array
  * the size of the array
  * the comparison function
where we use a function pointer to provide the comparison function
(is there any alternative to providing a comparison function via
the arg list?).

Write this function generalized bubblesort using this prototype:
int bs(int *x,int size,int (*compare)(int x,int y)) { ... }

Now write two comparisons:
int lt(int x,int y) { ... }
int gt(int x,int y) { ... }

And test your bubblesort with these
two comparison functions. One possible test program would look
like:
int main(void) {
   int i=0;
   int vals[10];
   for (i=0;i<10;i=i+1){
      vals[i]=100-i;
   }
   for (i=0;i<10;i++){
      printf("in[%d]=%d\n",i,vals[i]);
   }
```

```
    /* HERE: call bs() with the appropriate comparison function */
    for (i=0;i<10;i++){
       printf("out[%d]=%d\n",i,vals[i]);
    }
    return 0;
}
```

Submit bs.c which should contain bs(), lt() and gt().
No other files are needed for this; no helpers.
(This is about 30 lines of code: a 5-20 minute exercise that could
be asked on a test).

B. How classes are made; How classes can be faked...
C does not have the "class" functionality of Python. But generations
of C programmers have productively created complex pieces of software
without a class construct. The class is an extremely useful method
of organizing one's logic: how did C programmers manage to survive and
manage complexity?

There is no magic behind classes, and with the struct and the
function pointer one can go quite far.

In this assignment you are going to write an implementation of a
priority queue --- i.e., a heap --- in C that can be selected by the
user of the heap to be a max or min heap.

```
typedef struct {
   int *store;
   unsigned int size;
   unsigned int end;
   int (*compare)(int x,int y);
} intHeap_T;
```

The user will use this heap as follows:

```
...
intHeap_T heap;
heap.store=(int *)malloc(1000*sizeof(int))
heap.size=1000;
heap.end=0; /* empty heap condition; obey this spec */
heap.compare = lt; /* assuming lt was defined as in part A */
```

What kind of heap was created above? Min or Max? How would
you make the other kind?

Finish this heap, writing the:

```
int store(intHeap_T* heap,int value) { ... }
int retrieve(intHeap_T* heap,int *rvalue) { ... }
```

functions (return value is 0 for success, -1 for error) with
arguments that you can easily infer from the context above.

heap.c should contain the typedef and the two functions.
Submit it.

Make sure you test your heap to ensure it works.

C. Heap Sort

Read about heap sort and implement the function:

```
int hs(int *x,int size,int (*compare)(int x,int y)) { ... }
```

which uses the heap you created above to sort the array x.

You can control the type of sort (reverse-descending or descending)
by passing an appropriate compare function.

```
Submit this as hs.c; one file, like the above assignments.
This file should be self contained: all code you need should be IN the file
and not #include (except for stdio, stdlib, etc.). Do not #include your part B
heap but copy it into hs.c.


For all cases above:
the use of < or lt() will result in a descending sort, and a max heap
the use of > or gt() will result in an ascending sort, and a min heap.
This concludes the spec.
```