Programming with Arrays in C and MIPS Assembly Language

Next week, there will be no CS 237 lab meeting because of fall reading period. Therefore, we will pack two exciting exercises into this week's lab.

Faster than Quicksort?

In general, the fastest algorithm for sorting n values take $n \log n$ steps. However, if the values to be sorted are all known to fall within a reasonably small range, there is a linear algorithm called radix sort. The easiest way to understand radix sort is to think of sorting multi-digit numbers. If the numbers fall in a pre-specified range, then we would know the maximum number of digits in any of the numbers. To sort, we could then first go through the list looking only at the lowest order digits and get the numbers sorted according to this digit. That is, all the numbers ending in 0 would go first followed by all the numbers that ended with 1, followed by all the numbers that ended with 2 and so on up to 9. Once this was done, we repeat the process looking only at the next to last digit in the number. We would make a list of numbers whose next to last digit is 0, being careful to keep them in order according to their last digit (i.e., all of the numbers that ended with 00 would be before the 01s which would be before the 02s, etc.) and similar lists for numbers whose last digit is 1, all the way up to 9.

If we repeat this process for every digit (assuming all short numbers are padded with leading 0s), the numbers will eventually be sorted. Each pass takes time n. The number of passes is determined by the size of the numbers not the number of numbers. If the largest number has k digits, the whole process take kn steps, so it is still linear in the number of input values.

We would like you to implement this algorithm in C. Your program should be stored in a file named radixSort.c. The input to your program will be a list of positive integer values (i.e. positive numbers that can be represented in 32-bit twos complement form). You should read these values from standard input using scanf. Its output should be a list of the input values in increasing order printed one per line.

Your program should not sort the numbers by decimal digits as suggested in our explanation of the algorithm. Instead, it should sort according to either octal or hexadecimal digits. It should be possible to switch back and forth between octal and hexadecimal by simply changing one or two #defined constants in the program.

Like Quicksort, your program will work by copying the values it is given as input back and forth between two arrays. Your main program should read them into one array which it passes as a parameter to your radix sort function. The radix sort function will have a local work array of the same size of the input array. On each pass, the values will be copied from one array to the other. If necessary, a final pass should copy them back to the input array.

To make this possible, each pass will actually require two sub-passes. The first sub-pass will determine how many times each possible digit occurs in the digit position being processed. The second sub-pass will use this information to know where it should place each number encountered as it scans the array. For example, if on the first sub-pass you count 9 numbers with 0 in the nth digit position, 4 numbers with 1 in the nth position, and 6 numbers with 2 in the nth position, then after the nth pass is completed, array elements 0 through 8 should hold numbers with 0 in the nth position, array elements 9 through 12 should hold numbers with 1 in the nth position, array elements 13 through 18 should hold numbers with 2 in the nth position, and so on.

Hint: You may find that C's shift operations are useful in this program.

Heaping it on

For the second exercise, we would like you to translate the C code that implements heap sort shown below (and available as link on the course web page) into a MIPS assembler file named heapSort.asm. This program expects its input to be provided through command line arguments rather than typed in as standard input.

```
#include <stdio.h>

// The values to be sorted
int values[20];

// How many values are in the values array
int valc;

// Swap two parameters passed by reference
void swap( int *x, int * y ) {
  int temp;

  temp = *x;
  *x = *y;
  *y = temp;
}

void heapify( int values[], int rootIndex, int heapSize ) {
  int smallestChild = 2*rootIndex + 1;

  while ( smallestChild < heapSize ) {
    int sibling = smallestChild + 1;
    if ( sibling < heapSize &&
         values[ sibling ] < values[ smallestChild ] ) {
         smallestChild = sibling;
    }

    if ( values[rootIndex] < values[smallestChild] ) {
      return;

    } else {
      swap( &values[rootIndex], &values[smallestChild] );
      rootIndex = smallestChild;
      smallestChild = 2*rootIndex + 1;
    }

  }
}

// Takes up to 20 numeric arguments and sorts them in descending order
int main( int argc, char * argv[] ) {
  int p;

  for ( p = 1; p < argc; p++ ) {
    values[p-1] = atoi( argv[p] );
  }

  valc = argc - 1;

  // HEAP SORT!

  for ( p = valc - 1; 0 <= p; p-- ) {
    heapify( values, p, valc );
  }

  int remaining = valc;
  while ( remaining > 0 ) {
    remaining--;
    swap( & values[0], &values[remaining] );
    heapify( values, 0, remaining );
  }

  // Print the result
  for ( p = 0; p < valc; p++ ) {
    printf( "%d ", values[p] );
  }

  printf( "\n" );
}
```

Submitting Your Work

Within a terminal window, use the cd command to make the directory containing your .c and .asm files your current working directory. Then type the command:

```
turnin -c 237 heapSort.asm radixSort.c
```

Respond to the prompts appropriately and your work should be submitted.