CS314 - Operating Systems Homework 0 (Fall '23)

1 Administrivia

This is an introductory assignment. Unix utilities will be graded (100 points), the rest will receive full credit (100 more points) for a reasonable attempt.

To get the most out of these exercises (and to receive credit for your work):

1. Code everything in C, and ensure it compiles and runs on os.cs.siue.edu. Where function prototypes are given, do not modify them (some are written to guide you to a particular solution).
2. Create a makefile which builds all solutions by default (e.g. simply typing `make`), but also allows for each solution to be built separately (look at PHONY).
3. Place your work in a directory called `hw0`, containing three subdirectories: `utilities`, `easy`, and `bits`, with the programs given below placed into the appropriate directory.

1.1 Unix Utilities

1. (30 pts) See the man page for the `cp` utility. Used for copying files, it a program essential to every system. Ensure that your solution works with both text and binary files (do not use any of the printf family of functions to implement, instead, look at read/write or fread/fwrite). You can use `diff` to check for correctness.
2. (40 pts) See the man page for the `wc` utility. It will be present in any UNIX-like system distribution. Try the wc utility out on some input to familiarize yourself with its default output, then write a `wc` utility of your own that is invoked and produces similar output, counting the number of characters, words, and lines present in the input file. You do not have to implement support for options supported by the real `wc` utility, just the default behavior for one or more files specified on the command line. If no file is specified, input is taken from stdin until EOF appears (ctrl+d).

   Example:

```
$ ./wc wc.c
      34      85     455 wc.c
$ ./wc wc.c a.out
      34      85     455 wc.c
       6      40    8544 a.out
      40     125    8999 total
$ ./wc
this should work as well!
       1       5      26
```

3. (30 pts) See the man page for the `cat` utility. It reads files sequentially, writing out their contents to stdout. The files are read and written out to stdout in the order they appear in in the invocation (e.q. "cat file1 file2" prints contents of file1 then file2), printing an error message when it is unable to read a file or a file is missing and moving on to the next one in the list. Implement the basic functionality of cat (for no input files, where cat reads input from stdin, and for up to 5 files). Also add support for the "-b" flag, which numbers each line of output starting at 1.

1.2 Ten Easy Pieces

1. Write a function `set(int* a, int val)` that sets the value of an integer a declared in the calling function, to the value `val`. Print the value of the integer from the calling function both prior to and after calling the `set` function.
2. Write the function `swapInts(int*, int*)` that swaps the values of two integers. From the calling function, print the values of two pointers before and after the swap.
3. Write the function `swapPtrs(int**, int**)` that swaps two pointers. From the calling function, print the values of the two integers before and after the swap.
4. Implement a singly-linked list of integers. Write the function `findNthElement(struct listnode* head, int N)` which returns the Nth element of the list. Print the contents of your list and the value of the Nth element on separate lines.
5. Implement a singly-linked list of integers. Write the function `removeNthElement(struct listnode** head, int N)` which removes the Nth element of the list. Print the contents of the list before and after removal of the Nth element.
6. Implement a singly-linked list of integers. Write the function `findNthToLastElement(struct listnode* head, int N)` which returns the Nth element counting from the END of the list. Print the contents of the list as well as the Nth to last element on separate lines.
7. Implement a doubly-linked list. Write the function `reverse(struct listnode** head)` which reverses the list by swapping the previous and next pointers of each node. Print the contents of the list before and after the reversal.
8. Using your doubly-linked list and reverse function, write a function `isPalyndrome(struct listnode* head)` that returns 1 if the string encoded in your list is a palyndrome and 0 if it is not.
9. Construct a binary search tree (BST) that stores integers (i.e. the descendants to the left of a node are less than or equal to the node and the descendants to the right of the node are greater than or equal to the node). Insert integers 100, 75, 50, 125, 25, 150 into the tree. Write the function `preorderPrint(treenode*)` that uses recursion to perform a preorder traversal of the tree and print the values.
10. See previous problem. Write the function `nr_preorder_print(treenode*)` that performs a preorder traversal of the tree WITHOUT recursion. Instead, implement a stack (singly-linked list manipulated with push(...)/pop(...) functions) to aid with the traversal.

1.3 Four More Bits

1. Write the function `bitcount(unsigned x)` that returns the number of 1-bits in the unsigned integer argument x.

2. Write the function `invert(unsigned int x, int p, int n)` that returns x with the n bits that begin in position p inverted, leaving the others unchanged.

   Example:

   ```
   x = 0000 1111 1010 1010 1010
   p = 7
   n = 5
   y = invert(x, p, n) = 0000 1110 0101 1010 1010
                                 ^ ^^^^
   ```

3. Write the function `rightRot(int x, int n)` that returns the value of the integer x rotated to the right by n bit positions.

   Example:

   ```
   x = 0000 0000 0011 1100
   y = rightRot(x, 4) = 1100 0000 0000 0011
   ```

4. Write the function `setBits(int x, int p, int n, int y)` that returns x with the n bits that begin at position p set to the rightmost n bits of y. All other bits should remain unchanged.

   Example:

   ```
   x = 1010 1010 1010 1010
   y = 1100 1100 1100 1100
   z = setBits(x, 2, 4, y) = 1011 0010 1010 1010
                               ^^ ^^
   ```

2 What to Turn In

Create a compressed tarball (see `man tar`) or zip file containing just your homework directory and its contents, e.g. `hw0`, and its contents. Name the tarball `hw0.tgz`. Transfer the tarball off of the server (see `man scp`) and submit it on Moodle.