CS262, Project 3:
Radix Sort with Linked List

Due: Sunday, May 8, at 11:59 pm ET

Overview:

In this project you will write a C program to implement the **Radix Sort** Algorithm for sorting integer sequences. As part of this implementation, you will create functions for manipulating singly linked lists. The algorithms for implementing linked lists will be discussed in your classes. Therefore, this description does not list implementation details of the required functions.

Radix Sort:

For this sort, you choose a number of buckets **B**. This program will use B = 10 - the number of numerals used in a decimal representation of a value. **N** will be the exponent for a power of 10 corresponding to the most significant digit in any number in the sequence $S_0 = \{s_0, s_1, \ldots, s_m\}$ to be sorted. For example, if the largest value in the list has four digits (e.g. 3871), then **N** = 3, since 103 is a four digit number.

Radix sort distributes the elements in the buckets by digits, starting from the least significant to most significant. In the first pass the numbers are placed into buckets based on the value of the least significant digit. After that, a new sequence S_1 is created by stitching the bucket lists in order from the 0^{th} to 9^{th}. Next, the elements of S_1 are distributed in the buckets by the value of second digit (the tens place) and so on, up to the max number of digits.

```
Create 10 buckets/lists
Generate random values and place them in an unsorted linked list
Print the unsorted list
Determine N
for n = 0 to N do
        for all numbers in sequence Sn
                put the number in the bucket corresponding to the n-place digit
        Stitch the bucket lists together from 0th to 9th to create Sn+1
Print the sorted list
```

Note that when the algorithm completes, the only non-empty bucket is bucket 0.

Example. Let's sort the sequence $S_0 = \{32, 100, 11, 554, 626, 122, 87, 963, 265, 108, 9\}$.
We start by distributing elements of S_0 by the value of 0-place digit (the *one's* place):

bucket 0: 100
bucket 1: 11
bucket 2: 32, 122
bucket 3: 963
bucket 4: 554
bucket 5: 265
bucket 6: 626
bucket 7: 87
bucket 8: 108
bucket 9: 9

Stitch the bucket lists to create S_1 = {100, 11, 32, 122, 963, 554, 265, 626, 87, 108, 9}.
Distribute elements of S_1 by the value of 1-place digit (the *ten's* place):

bucket 0: 100, 108, 9
bucket 1: 11
bucket 2: 122, 626
bucket 3: 32
bucket 4:
bucket 5: 554
bucket 6: 963, 265
bucket 7:
bucket 8: 87
bucket 9:

Stitch the bucket lists to create S_2 = {100, 108, 9, 11, 122, 626, 32, 554, 963, 265, 87}.
Distribute elements of S_2 by the value of 2-place digit (the *hundred's* place):

bucket 0: 9, 11, 32, 87
bucket 1: 100, 108, 122
bucket 2: 265
bucket 3:
bucket 4:
bucket 5: 554
bucket 6: 626
bucket 7:
bucket 8:
bucket 9: 963

Stitch the bucket to create S_3 = {9, 11, 32, 87, 100, 108, 122, 265, 554, 626, 963}. The list is sorted.

Detailed requirements for linked lists implementation:

To represent a node in a linked list, use a structure:

```
typedef struct Node{
  DataType      data;
  struct Node   *next;
}ListNode;
```

1. Write a function to make a new list.
   Your function should create a dummy head node to represent an empty list.
   /* returns a head of a new empty list */
   *ListNode *newList(void);*

2. Write functions to insert elements into a list and to remove elements from a list (Note: You may have additional functions, or different function prototypes than those shown)
   /* remove the node after prev from the list, and returns a pointer to the removed node */
   *ListNode *removeNode(ListNode * prev);*
   /* inserts a new node with data field data after prev and returns a pointer to the new node */
   *ListNode *insertNode(ListNode *prev, DataType data);*

3. Write functions to count the number of elements in a list without the head and to print the list
   *int length(ListNode *head);* /* number of elements in the list */
   *void printList(ListNode *head);* /* print the data fields for the entire list */

   Note that *printList* will print the linked list implemented to support 2nd part of your project.

- To make radix sort more efficient you will find it useful to implement a function
  *ListNode insert at tail(ListNode *tail, DataType data)*
  The function should attach a new node to the tail and make that node the new tail of the list.
- You should also implement a *deleteList(ListNode *head)* function which can be used to delete an entire list.

Detailed requirements for linked radix sort implementation:

You will need to write the functions for the linked lists implementation and the main `p3_<username>_<labsection>.c` program that will generate **N** random integers in the range `[low, high]` (inclusive). The program should take 4 arguments: a seed for the random number generator (`Seed`), the number of values to sort (`NumVals`), and low (`low`) and high values (`high`) to denote the range of values:

You have to free all memory before exiting the program—i.e. delete all linked lists. You will run *valgrind* as part of your submission typescript so use it during development and debugging to ensure there are no memory leaks or dangling pointers.

Some hints:

You will need an array of ten link list heads and an array of the ten corresponding tails (should you decide to implement tail pointers). You may also find it useful to maintain some auxiliary linked list for intermediate steps such as stitching the sublists, etc.

Makefile:

Your program must compile with no warnings using the following compile flags:

```
CFLAGS = -Wall -g -std=c89 -D_XOPEN_SOURCE=700 -pedantic-errors
```

Submitting:

1. On zeus, create a directory named `p3_<username>_<labsection>`
   Copy your source file and *Makefile* to this directory.
2. Create a typescript with the following content:
   a. Show that you are on zeus,
   b. Show a listing of your directory
   c. Show your source code
   d. Compile the code using the *Makefile*.
   e. Run your code using *valgrind*
      `valgrind --leak-check=yes p3 Seed NumVals low high`

Note that you need to test your program with different values of the input parameters. We will test it with something like N ≥ 100, and low > 0 and high > low. You should print the lists obtained after each important step, such as distributing values over lists by the currently used digit value and the list obtained after stitching the sublists together.

3. Be sure your directory ONLY contains the *source file*, *typescript* and *Makefile*
4. Change to the parent directory and create a tarfile of your project directory.
   Name this tarfile `p3_<username>_<labsection>.tar`
5. Submit this tarfile to Blackboard no later than Due Date.

Congratulations! You have completed your Project