COP4338 Systems Programming

Programming Assignment 3:
Structs in C

Knight Foundation School of Computing & Info. Sciences
Florida International University

In this assignment, you are asked to write a program that gets a series of string chars separated by space and followed by EOF from user through standard input, alphabetically sorts them using bucket sort, and prints the sorted strings out on the console.

Bucket Sort

Bucket sort, or bin sort, is a sorting algorithm that works by distributing the elements into a number of buckets. Each bucket is sorted individually. To sort each bucket in this assignment, you will use the *insertion sort* algorithm. Also, to compare strings with one another, you need to use function *strcasecmp* from string.h library as this sorting machine is not case sensitive.

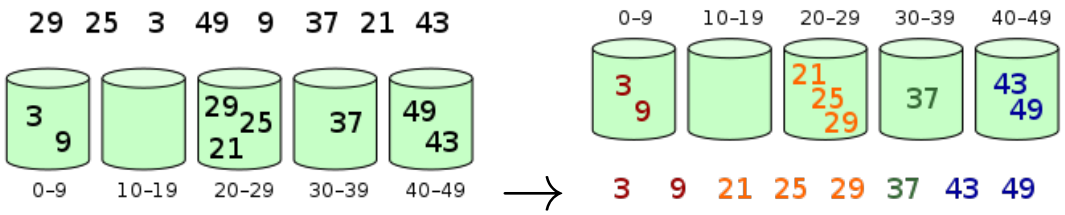

Figure 1: Example of bucket sort for sorting numbers 29, 25, 3, 49, 9, 37, 21, 43. Numbers are distributed among bins (left image). Then, elements are sorted within each bin (right image).

Bucket Sort's Data Structure

In order to implement bucket sort algorithm for strings, you need to have an array of buckets such that *bucket* is a C structure defined in the following way:

```
struct bucket{
    char* min_word;
    struct node* head;
};
```

and *node* is another C structure defined in the following way:

```
struct node{
    char* string;
    struct node* next;
};
```

The array of buckets must be initialized based on the command-line arguments. For example, if the program is executed using the command "./bucketsort apple carrot kiwi pineapple watermelon", then the buckets array should be constructed in the following way:

Bucket Index	min word	head
0	""	NULL
1	"apple"	NULL
2	"carrot"	NULL
3	"kiwi"	NULL
4	"pineapple"	NULL
5	"watermellon"	NULL

Each bucket contains two members:

- *char* min_word*: determines the minimum string that can be stored in each bucket (including the min_word itself); i.e. any string that comes before min_word in alphabetical order cannot be stored in this bucket. This word also serves as the upperbound string for the previous bucket; i.e. any string that is equal to or comes after min_word in the alphabetical order cannot be stored in the previous bucket. The value of min_word for the very first bucket is always the empty string.

- *struct node* head*: points to a singly linked list which is a chain of nodes containing the strings in each bucket. The list of words in each bucket is initially empty and *remains sorted* at all times.

How to Fill Buckets

Assume that user enters a series of string chars separated by space and followed by an EOF. Assume that no string contains tab or new line character. Every string of chars that user passes to the program must be stored in the format of a char string. To allocate memory for storing a char string in memory, you can use expression *char* word = (char *)malloc(strlen(input_string)+1);* in which *input_string* is a *char ** pointing to the array of letters given by the user, *strlen* and *malloc* are functions defined in *stdlib.h* library (note that *+1* in the mentioned expression allocates an extra memory cell for \0 that specifies the end of a char string).

After allocating enough memory cells for each input string and copying the string into them (*strcpy(word, input_string)*), you need to allocate memory for storing a variable of type *struct node* containing a *char ** (same as the mentioned char* word) and a *struct node **

pointing to the next node in the linked list of the appropriate bucket. The new node must be stored in a way that the linked list remains sorted. Therefore, it should be placed:

- after all nodes storing words that come before the new word alphabetically;
- AND before all nodes storing words that come after the new word alphabetically.

By doing this, you're gradually applying insertion sort to each bucket as the bucket is getting populated by input words.

A Simple Example

Your program must print out the sorted list of strings in an alphabetical order using *printf* function. In addition, it should specify which bucket each word was stored at. For example, if command-line arguments are "./bucketsort apple carrot kiwi pineapple watermelon" and stdin receives "This is a simple example with 8 words.", the output should be:

"bucket 0: 8 a
bucket 2: example is
bucket 4: simple This
bucket 5: with words."

As you see, each line of the output shows the index and content of a non-empty bucket. Please remember that the program expect to receive *at-least one* command-line argument (argc ≥ 2). Therefore, there must be at-least two buckets for the sorting algorithm.

Submission

You need to submit a *.zip* file compressing the makefile, C source file(s), and possibly any header file(s) related to the assignment (*.c* and *.h* files). Please **use "bucketsort" as the name of your program's executable file in the makefile**.