Assignment 4

Due Wednesday, Nov. 11, 2015

As in Assignment 3, you are to write a C program that extracts all words from a piece of text. This time, however, you are to store the words in a **hash table** as discussed in class (see Lecture Notes 8). The hash table should use chaining to resolve collisions. That is, words that hash to the same "bucket" should be stored as a singly linked list of nodes for that bucket.

A *word* is a contiguous sequence of characters delimited by any of the following punctuation characters: the space character, period (`.`), comma (`,`), semicolon (`;`), colon (`:`), exclamation point (`!`), double quote (`"`), question mark (`?`), and newline character (`\n`). Note that the single quote or apostrophe (`'`) is not considered a delimiter. As a result, the following are considered single words: **`sheep's 'Tis you're Charles'`**

Two words are considered the same if they consist of exactly the same sequence of characters, *ignoring case*. For example, "**`Boy`**" and "**`boy`**" are considered the same word. Words should be stored in *lower-case* in the hash table.

Each word in the hash table should correspond to a unique node in the linked list for the bucket it hashes to. Each node should be a structure consisting of the following members:

- **`word`:** a string representing the word
- **`count`:** the number of times that **`word`** appears in the text (ignoring case). That is, the first time **`word`** is encountered in the text, insert it into the hash table by creating a new node for it and initialize its **`count`** to 1. For each subsequent occurrence of **`word`** in the text, simply increment its node's **`count`** by 1.
- **`next`:** a pointer to the next node in the list (if it exists)

A sample type definition for a node is shown below (which is used in the subsequence):

```
struct NodeType {
   char *word;
   int count;
   struct NodeType *next;
};
typedef struct NodeType Node;
```

The size (i.e., number of buckets) of the hash table, say **`htsize`**, should be specified as a command-line argument. Make **`htsize`** a global variable. Upon reading **`htsize`**, allocate *from the heap* a hash table with **`htsize`** buckets, each initially empty.

Your program should read several lines of text stored in a file and redirected to the standard input **`stdin`**. It should then parse the lines into individual words and store each word – *in lower-case* – into the hash table. *All nodes created in the hash table should be allocated from the heap.* After reading the entire text, print all the **`[word,count]`** tuples stored in the hash table (see below). Finally, destroy the hash table by freeing all the heap space allocated to it.

As an example, suppose that the following text is stored in the file **`input.txt`**.

```
Little Boy Blue,
Come blow your horn,
The sheep's in the meadow,
The cow's in the corn;
Where is that boy
Who looks after the sheep?
Under the haystack
Fast asleep.
Will you wake him?
Oh no, not I,
For if I do
He will surely cry.
```

The following illustrates what your program should output for the above text:

```
clamshell:~> ./hashtable < input.txt
ERROR: Usage: ./hashtable table_size
clamshell:~> ./hashtable 23 < input.txt
HT[0]: [under, 1]  [oh, 1]
HT[1]: [that, 1]
HT[2]: [meadow, 1]
HT[3]: [cow's, 1]  [for, 1]  [if, 1]
HT[4]: [sheep's, 1]  [where, 1]  [no, 1]  [not, 1]
HT[5]: [wake, 1]  [cry, 1]
HT[6]:
HT[7]:
HT[8]: [surely, 1]
HT[9]: [blue, 1]  [blow, 1]  [looks, 1]
HT[10]:
HT[11]: [your, 1]  [in, 2]
HT[12]:
HT[13]: [little, 1]  [haystack, 1]  [i, 2]
HT[14]: [will, 2]
HT[15]:
HT[16]: [horn, 1]  [is, 1]  [fast, 1]  [asleep, 1]
HT[17]: [sheep, 1]
HT[18]: [you, 1]
HT[19]: [come, 1]  [the, 6]
HT[20]: [corn, 1]  [who, 1]  [him, 1]  [do, 1]
HT[21]: [after, 1]
HT[22]: [boy, 2]  [he, 1]
clamshell:~>
```

```
#define HASH_MULTIPLIER 65599
int htsize;
...
unsigned int hash(const char *str)
{
   int i;
   unsigned int h = 0U;

   for (i = 0; str[i] != '\0'; i++)
      h = h * HASH_MULTIPLIER + (unsigned char) str[i];

   return h % htsize;
}
```

Let **Table** be a pointer to the heap-allocated hash table declared as follows:

```
Node **Table;
```

Your program should contain at least the following functions:

- **unsigned int hash(const char *str):** compute and return the bucket to which the string **str** hashes. You should use the hash function discussed in class, as given above.

- **Node **ht_create(void):** create a heap-allocated hash table with **htsize** buckets, initially empty, and return a pointer to it. If the table cannot be allocated, issue an error message on **stderr** and terminate program.

- **int ht_insert(Node **Table, const char *word):** insert **word** in lower-case into the hash table **Table**. If **word** is not in **Table**, insert a new node for the word in the bucket to which it hashes to, and initialize its count to 1. The node should be inserted at the end of the list for the bucket. If **word** is already in **Table**, increment its count by 1. Return 1 on success, else return 0.

- **void ht_print(Node **Table):** print all words stored in the hash table **Table**. Specifically, iterate over the buckets of the hash table and print the **[word,count]** tuples hashed to each bucket.

- **void ht_destroy(Node **Table):** destroy the hash table **Table** by freeing all the space allocated to the table.

Test your program exhaustively. Specifically, it should detect missing command-line arguments, extract the correct tokens/words from the input text, and handle empty input. Check the return values of library function calls especially **malloc/calloc**. Use the **assert** facility where appropriate to check for pre- or post-conditions. Use **valgrind** to check for memory leaks and correct them.

Submitting Your Assignment

- Upload your C source code to the sakai website. Write your full name clearly in comments at the top of your program.
- Be sure to properly comment your code. Points will be taken off for improperly or insufficiently commented code.
- Be sure your code has no compilation errors. Compiler errors will prevent your submission from being graded.
- This assignment is due by 11:55 pm on Wednesday, e11, 2015. **Absolutely no late assignments will be accepted.**