CSI 402 – Systems Programming – Spring 2013

Programming Assignment I

Date given: Feb. 5, 2013 **Due date:** Feb. 15, 2013
Weightage: 5%

The regular deadline for this assignment is **11 PM, Friday, February 15, 2013**. With lateness penalty, the program will be accepted until **11 PM, Sunday, Feb. 17, 2013**. The assignment will *not* be accepted after 11 PM on Sunday, February 17, 2013.

Very important: Your source program must consist of three or more C files and you must also have a `makefile`. (Additional information regarding this requirement is given later in this handout.) The C files (with extension "`.c`"), header files (with extension "`.h`") and the `makefile` must be submitted together using the `turnin-csi402` command. Instructions for using `turnin-csi402` and additional specifications for the `makefile` will be included in the `README` file for this assignment.

The total grade for the assignment is 100 points, with 85 points for correctness and 15 points for structure and documentation.

The purpose of this assignment is to experimentally compare the performances of the two hash functions given on pages 3 and 4 of this handout. (Necessary information about hashing will be presented in class.) The source files for these functions are available in `~csi402/public/prog1`. Note that you *should not* modify these functions.

The executable version of your program must be named `p1`. (Your `makefile` must ensure this.) The program will be executed by a Unix command line of the following form:

`p1` *infile* *outfile*

The command line parameters *infile* and *outfile* represent the names of the input and output files respectively. Both of these are text files. The first line of the input file has an integer that specifies the size of the hash table. Each subsequent line of the input file has a symbol that must be inserted into the hash table. You may assume the following. (There is no need to check these conditions in your program.)

1. The size of the hash table will be at least 1 and at most 10,000.
2. There are no blank lines in the input file.
3. Each symbol has at least one character and at most 15 characters. Each character of a symbol is an upper or lower case letter, a digit or underscore (`'_'`).
4. The symbols in the input file are all *distinct*.

For each hash function, your program should insert the symbols in the input file into the hash table using the chaining method to resolve collisions. After inserting all the symbols, your program should compute and print the following information to the output file.

1. Size of the hash table.
2. Number of non-empty lists in the hash table.
3. Maximum number of entries in a list.

4. Minimum number of entries in a non-empty list.

5. Average size of a non-empty list. (This is the ratio of the number of symbols in the input file to the number of non-empty lists. In general, this average value is a *real number*.)

Your program must detect the following errors. In each case, your program should produce a suitable error message to `stderr` and stop.

(1) The number of command line arguments is not equal to three.

(2) The input or the output file specified on the command line cannot be opened.

Structural requirements: Your submission must have *at least three* C source files, zero or more header files and a `makefile`. Two of the C source files must have the hash functions used by your program. (The C source files for these two functions are in `~csi402/public/prog1`. You should copy these source files to your directory.) The other C source file(s) must have the `main` function and other functions (e.g. functions that compute the required statistical values) that you may need.

Information about `README` file: The `README` file for this assignment will will be available by 10 PM on Saturday, February 9, 2013. The name of the file will be `prog1.README` and it will be in the directory `~csi402/public/prog1` on `itsunix.albany.edu`. (Note that this directory also contains the C source files for the two hash functions.)

Source code for the first hash function: The source code for this function is available as file `hash_one.c` in the directory `~csi402/public/prog1`.

```
int  hash_example_one (char *s,  int T) {

   /* The parameter s represents the symbol to be hashed and  */
   /* the parameter T represents the size of the hash table.  */
   /* The function returns the hash value for the symbol s.   */

   /* String s is assumed to be terminated with '\0'.         */
   /* It is also assumed that T is at least 2. The returned   */
   /* hash value is an integer in the range 0 to T-1.         */

   /* The function computes the hash value using arithmetic   */
   /* based on powers of the BASE value defined below.        */

   #define  BASE   127

   int h = 0;     /* Will hold the hash value at the end. */
   int temp;      /* Temporary.                           */

   /* The hash value is computed in the for loop below. */
   for (;  *s != 0;  s++) {
       temp = (BASE * h + *s);
       if (temp < 0) temp = -temp;
       h = temp % T;
   }

   /* The hash value computation is complete. So, */
   return h;

} /* End of hash_example_one */
```

(over)

Source code for the second hash function: The source code for this function is available as file `hash_two.c` in the directory `~csi402/public/prog1`.

```
int  hash_example_two (char *s,  int T) {

   /* The parameter s represents the symbol to be hashed and  */
   /* the parameter T represents the size of the hash table.  */
   /* The function returns the hash value for the symbol s.   */

   /* String s is assumed to be terminated with '\0'.         */
   /* It is also assumed that T is at least 2. The returned   */
   /* hash value is an integer in the range 0 to T-1.         */

   /* The function computes the hash value using bitwise      */
   /* operations (namely left shift and exclusive or).        */

   #define  SHIFT_AMOUNT   5

   int h = 0;     /* Will hold the hash value at the end. */

   /* The hash value is computed in the loop below. */

   for (;  *s != 0;  s++)
      h = (h << SHIFT_AMOUNT) ^ (*s) ^ h;

   /* If the resulting hash value is negative, change it  */
   /* to a positive value before taking the remainder     */
   /* modulo T.                                           */

   if (h < 0)
      h = -h;

   return  (h % T);  /* The returned hash value. */

} /* End of hash_example_two */
```