CSI 402 – Systems Programming – Handout 6.1
Examples of Hash Functions

Note: This handout provides two examples for hash functions that are useful in practice. In the C functions given below, the parameter **s** represents the string for which the hash value needs to be computed and the parameter **T** represents the size of the hash table.

The first example shows a hash function that uses arithmetic operations. The second example shows a hash function that uses bitwise operations.

Example 1: This hash function is from the following reference: R. Sedgwick, *Algorithms in C++* (third edition), Parts 1-4, Addison-Wesley, Boston, MA, 1998.

Let $s_{k-1}s_{k-2}\ldots s_1s_0$ represent a string consisting of k ASCII characters. Let $A(s_i)$ denote the ASCII value of the character s_i, $0 \le i \le k-1$. The hash function (`hash_example_one`) presented below computes

$$\left[\sum_{i=0}^{k-1}(A(s_i) \times 127^i)\right] \bmod T$$

where "mod" represents the remainder operation (the '`%`' operator in C) and T represents the size of the hash table. (In the following C code, we use a simple comparison to ensure that the value before computing the mod operation is non-negative.)

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

   #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 */
```

Example 2: This hash function is from the following reference: M. A. Weiss, *Algorithms, Data Structures and Problem Solving with C++*, Addison-Wesley, Menlo Park, CA, 1996.

There doesn't appear to be a simple explanation for value computed by this hash function. The main goals are to try to make the hash value depend all the characters in the string and to keep the computation time as small as possible. So, the function uses bitwise operations (left shift and exclusive or) and only one computation of the remainder.

As in Example 1, we use a simple comparison to ensure that the value before computing the mod operation is non-negative.

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

  #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 */
```