COMP2321—DATA STRUCTURES

Hashing Strings

Dr. Radi Jarrar
Department of Computer Science
Birzeit University

Hashing Strings (1)

- If the key is String, then the hash function can be chosen in one of the following techniques:

1. Add up the ASCII values of the character in the String.
   $key = (ASCII(a_0) + ASCII(a_1) + ASCII(a_2) + \ldots\ ) \ \%\ (HashSize)$

- Though this approach is simple to implement and quick, it has the following problems:

a. Words may have the same ASCII sum (e.g., net, ten)
b. If the table size is large, the function will not distribute keys well.

Hashing Strings (1)

```
int hash(char* key, int TableSize){
    int hashValue = 0;

    while( *key != '\0' ){
        hashValue += *key++;
    return (hashValue % TableSize);
}
```

Hashing Strings (2)

- Another Hash function assumes that the key has at least two characters plus the NULL terminator. The value 27 represents the number of letters in the English Alphabet, plus the blank, and $729 = 27^2$. This function examines the first three characters only. A problem with this approach is that there is no random distribution of keys.

$$key_1 = a_0 * 27^0 + a_1 * 27^1 + a_2 * 27^2$$
$$key_2 = a_0 * 27^0 + a_1 * 27^1 + a_2 * 27^2$$

Hashing Strings (2)

```
return ( (int)key[0] + (int)key[1] * 27 +
(int)key[2] * 729 ) % TableSize;
```

Hashing Strings (3)

Hash function that involves all characters in the key and can generally be expected to distribute the keys well.
It computes as

$$\sum_{i=0}^{KeySize-1} Key[KeySize - i - 1] . 32^{i}$$

which is a polynomial of 32 (a factor of 2) instead of 27 because it is faster to process than 27.

* 32 is bit shifting by 5

Hashing Strings (3)

```
int hash(char* key, int TableSize){
    int hashValue = 0;

    while( *key != '\0' ){
        hashValue = ( hashValue << 5 ) +
key++;

    return (hashValue % TableSize);
}
```

Hashing Strings (3)

- << is left-shifting, which is equivalent to multiplying a number by 2.

  Example: 8 in Binary is 00001000. Shift '8' two places to the left will give:
  8 << 2 → 00100000 (which is 32). It is the same as 8 * 2^2 = 32.
  Everything is moved to the left and zeros are added as paddings.

Hashing

- Note that the TableSize has to be a prime number.