Written Homework #3
CS 163: Data Structures

Alves Silva, Otavio Augusto
PSU ID: 902840168

1. Table Abstractions

a. Describe the difference between a list and a table

A list is form to store elements or data in an orderly, but the order is not determined by the list itself, but by its client. For example, an array of characters can be represented by a list of characters where any character has a specific index. A table otherwise has a function, which is called hash function, to calculate a index of this table using a key.

b. Compare and contrast three collision resolution techniques

In the open addressing approach resolution we have that in the moment that an attempt to insert an element into the table is already occupied, we have to look for another open space to insert this element. Therefore, we have some techniques, which were linear probing, quadratic probing, and double hashing, to execute this approach and resolve this problem.

1. **Linear probing:** It is the simple technique, which is basically searching for another empty space in the hash table starting from the original hash location. Therefore, just improving the table index, after the hash function call. However, this technique has some problems to remove an element because to find this specific element can be done incorrectly and the primary clustering is another common problem of this technique.
2. **Quadratic probing:** It has almost the same technique of the linear probing, but the improving of the table index in a squared way. With this the primary clustering can be resolved. However, this technique causes the secondary clustering, which is the delay of the collisions.
3. **Double hashing:** This technique drastically reduces the number of clustering because the linear and quadratic probing are key independent whereas the double hashing uses the key. Therefore, the probe still search the hash table in a linear order, but there is a second hash function that will determine the size of steps that will be taken.

c. Design a hash function for a table of keywords

The following hash function can create an amount of 777(0 – 776) keys for a hash table using an array of characters, which was considered a description in this approach. However, we don’t have an algorithm to provide an away to avoid collisions yet. The code of this function can be seen below.

```
int hash_function(char * desc)
{
     int key;
     int tableSize = 777;      // A bigger prime number
                               // to afford thousands of keys

     char * to_key = new char[strlen(desc)+1];       // Deep copy of a
     strcpy(to_key, desc);                           // specific description


     // Each character has a specific decimal value
     // The description will have a specific decimal value
     char alphabet[26] =
     {'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','
     u','v','w','x','y','z'};
     int  alphabet_values[26] =
     {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26};

     int len_alphabet = 25;
     int len = strlen(to_key)+1;
     int value = 0;

     // Loop to check each character in the description
     // Sum each value in to the value
     for(int x = 0; x < len; x++)
     {
        for(int y = 0; y < len_alphabet; y++ )
        {
               if(array[x] == alphabet[y])
                      value += alphabet_values[y];
        }
     }

     // The key is calculated with the modulo
     // to make shore that the key will not
     // be out of the range of the table
     key = value % tableSize;

     return key;
}
```

2. Ethics

The three rules that you would expect of my programmers are:

- The ADT, which the programmers will create, must protect the user's sensitive information.
- The ADT must have a "wall" from the client program as an away to protect the user's information.
- The programmers must use a privilege system in the moment that any user's information is requested to be another away to protect this information.

3. Linux

a. **Wall compiler flag**: It is an away to help the programmer find some mistakes that were made, but doesn't influence in the program flow and compilation at all. Therefore, it can help the programmer see all warnings that the program has in the moment that was compiled.

b. **Grep**: It is a command that allows finding a specific expression in the system by lines matching with the expression that was gave by argument. It is really useful to see if a specific file has an expression or not.