Hash Table Code

Alternative Big O notation:

$$
\begin{aligned}
O(1) &= O(\text{yeah}) \\
O(\log n) &= O(\text{nice}) \\
O(n) &= O(\text{ok}) \\
O(n \log n) &= O(\text{well}) \\
O(n^2) &= O(\text{my}) \\
O(2^n) &= O(\text{no}) \\
O(n!) &= O(\text{mg!})
\end{aligned}
$$

https://twitter.com/jwcarroll/status/1114576190247976960?s=11

1 Hash Tables

A **Hash function** transforms keys into an array index.

Enables fast lookup of information.

Collision Resolution

One problem with all hash functions is **collisions**. A collision is the result of two or more keys hashing to the same value (location).

Two common **collision resolution** strategies:

- Linear resolution
- Open hashing

1.1 Hash Table Code

```
/*  hash.h
 */

#ifndef HASH_H
#define HASH_H

struct nList     /*  table entry:  */
{
   char *name;          /*  defined name         */
   char *defn;          /*  replacement text     */
   struct nList *next;  /*  next entry in chain  */
};

typedef struct nList *NListPtr;

unsigned Hash( char *s );
NListPtr Lookup( char *s );
NListPtr Insert( char *name, char *defn );

void PrintHashTable();

#endif  /*  HASH_H  */
```

```
/* Hash.cpp
 *
 *  Hash table implementation from:
 *  Kernighan & Ritchie, The C Programming Language,
 *     Second Edition, Prentice-Hall, 1988.
 */

#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <string.h>

#include "hash.h"


const int HASH_TABLE_SIZE = 101;
static NListPtr hashTable[HASH_TABLE_SIZE];

    //  Prototypes
char *strdup( const char * );  //  in string.h, but....
```

```
/*  Hash
 *  Generate hash value for string s
 */

unsigned Hash( char *s )
{
    unsigned hashVal;

    for( hashVal = 0 ; *s != '\0' ; s++ )
        hashVal = *s + 31 * hashVal;

    return  hashVal % HASH_TABLE_SIZE;
}
```

```
/*  Lookup
 *  Look for s in hashTable
 */

NListPtr Lookup( char *s )
{
    NListPtr np;

    for( np = hashTable[Hash(s)] ; np != NULL ; np = np->next )
    {
        if( strcmp(s, np->name) == 0 )
            return np;    //  found
    }

    return NULL;          //  not found
}
```

```
/*  Insert
 *  Put (name, defn) in hash table
 */

NListPtr Insert( char *name, char *defn )
{
    unsigned hashVal;
    NListPtr np;

    if( (np = Lookup(name)) == NULL )  // not found
    {
        np = (NListPtr) malloc(sizeof(*np));
        if( np == NULL || (np->name = strdup(name)) == NULL )
            return NULL;
        hashVal = Hash(name);
        np->next = hashTable[hashVal];
        hashTable[hashVal] = np;
    }
    else
    {      //  remove previous definition
        free( (void *)np->defn );
    }

    if( (np->defn = strdup(defn)) == NULL )
        return NULL;

    return np;
}
```

```
/*  PrintHashTable
 *  Print the hash table contents
 */

void PrintHashTable()
{
    NListPtr np;

    cout << "Hash table contents:" << endl;
    cout << "--------------------\n" << endl;

    for( int i = 0 ; i < HASH_TABLE_SIZE ; i++ )
    {
        np = hashTable[i];
        while( np != NULL )
        {
             cout << setw(3) << i << ":    ";
             cout << np->name << ", " << np->defn;
             cout << endl;
             np = np->next;
        }
    }
}
```

```
/*  strdup
 *  Make a duplicate copy of s
 */

char *strdup( const char *s )
{
    char *p;

    p = (char *)malloc(strlen(s)+1);  /*  +1 for '\0'  */
    if( p != NULL )
        strcpy(p,s);

    return p;
}
```

```
/* TestHash.cpp
 *   Test the Hash table code.
 */

#include <iostream>
#include <cstdlib>

#include "hash.h"


int main()
{
      //  Put a few values in the table...
   (void)Insert( "One",  "1" );
   (void)Insert( "One",  "11" );
   (void)Insert( "Two",  "2" );
   (void)Insert( "Four", "4" );
   (void)Insert( "Five", "5" );
   (void)Insert( "Six",  "6" );
   (void)Insert( "Nine", "9" );

   (void)Insert( "Yes", "1" );
   (void)Insert( "YES", "1" );
   (void)Insert( "No",  "0" );
   (void)Insert( "NO",  "0" );

   PrintHashTable();

   return EXIT_SUCCESS;
}
```

```
Hash table contents:
--------------------

  4:    No, 0
 15:    Six, 6
 42:    Four, 4
 44:    One, 11
 51:    Five, 5
 73:    NO, 0
 74:    Nine, 9
 83:    YES, 1
 88:    Two, 2
 97:    Yes, 1
```