8 Searching Supplementary Information
Hashing Demo w/ Chaining to resolve Collisions

Source Code

In this example we implement a hash table in which we implement hashing and use chaining to resolve the collisions.

We use the same hash table size and hash function here as in the example so that you can see exactly how we got the example's results.

```
/*hashll.c: hashing demo w/ chaining to resolve collision*/
/* (c) 2001 by Ian Chai */
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

#define HASHTABLESIZE 7

struct LLNode{
  int key;
  struct LLNode *next;
};
struct LLNode *hashTable[HASHTABLESIZE];
void initializeHashTable(void);
struct LLNode *initLLNode(int val);
void insertLList(int val, struct LLNode **llist);
char inList(int val, struct LLNode *list);
void printList(struct LLNode *list);
int hashFunction(int key);
void insertHash(int val);
char inHash(int val);
void printHashTable(void);

int main(void){
  int value, what;
  initializeHashTable();
  printf("1\tInsert new value into hash table.\n");
  printf("2\tCheck if a value is in the hash table.\n");
  printf("3\tPrint the entire hash table.\n");
  printf("0\tQuit\n\n");
  do{
    printf("Input:\t");
    scanf("%d",&what);
    switch(what){
      case 1:
        printf("\t\t\tNew value? ");
        scanf("%d",&value);
        insertHash(value);
        break;
      case 2:
        printf("\t\t\tNew value? ");
        scanf("%d",&value);
        if (inHash(value))
          printf("Exists in table.\n\n");
        else
          printf("Not in table.\n\n");
        break;
      case 3:
        printHashTable();
        break;
    }
  } while(what);
  printf("Finished\n");
  return(0);
}
```

Functions for the hashing algorithm

```
void initializeHashTable(void){
  int i;
  for (i=0; i<HASHTABLESIZE; i++)
    hashTable[i] = NULL;
}
```

Every hash bin starts out as an empty linked list.

```
int hashFunction(int key){
  return(key % HASHTABLESIZE);
}
```

Just like in the example, our hash function is *key mod 7.*

```
void insertHash(int val){
  insertLList(val,&(hashTable[hashFunction(val)]));
}
```

The item goes into the list corresponding to that given by the hash function.

```
char inHash(int val){
  return(inList(val, hashTable[hashFunction(val)]));
}

void printHashTable(void){
  int i;
  for (i=0; i<HASHTABLESIZE; i++){
    printf("Hash bin %d: ",i);
    printList(hashTable[i]);
    printf("\n");
  }
}
```

Everything from here on down are just support functions for the linked lists and not part of the hashing algorithm itself.

```c
struct LLNode *initLLNode(int val){
  struct LLNode *temp;
  temp=(struct LLNode *)malloc(sizeof(struct LLNode));
  temp->key=val;
  temp->next=NULL;
  return(temp);
}
```

This is the same function from §3.2, only the data is integer rather than char.

```c
void insertLList(int val, struct LLNode **llist){
  struct LLNode *newnode = initLLNode(val);
  newnode->next = *llist;
  *llist = newnode;
}
```

This implementation doesn't bother to check to see if *val* has been already entered into the linked list, so you can get duplicates – the answers will still be right, but it could be less efficient as you might have to search through more nodes to get the answer.

```c
char inList(int val, struct LLNode *list){
  if (list==NULL)
    return(0);
  else{
    if (list->key == val)
      return(1);
    else
      return(inList(val, list->next));
  }
}
```

Returns 1 if *val* is in the list, otherwise returns 0.

```c
void printList(struct LLNode *list){
  if (list){
    printf("%d ",list->key);
    printList(list->next);
  }
}
```

Sample Output

```
1       Insert new value into hash table.
2       Check if a value is in the hash table.
3       Print the entire hash table.
0       Quit

Input:  1
                        New value? 12
Input:  1
                        New value? 15
Input:  1
```

```
                        New value? 21
Input:  1
                        New value? 36
Input:  1
                        New value? 84
Input:  1
                        New value? 96
Input:  3
Hash bin 0: 84 21
Hash bin 1: 36 15
Hash bin 2:
Hash bin 3:
Hash bin 4:
Hash bin 5: 96 12
Hash bin 6:
Input:  2
                        New value? 36
Exists in table.

Input:  2
                        New value? 12
Exists in table.

Input:  2
                        New value? 32
Not in table.

Input:  0
Finished
```