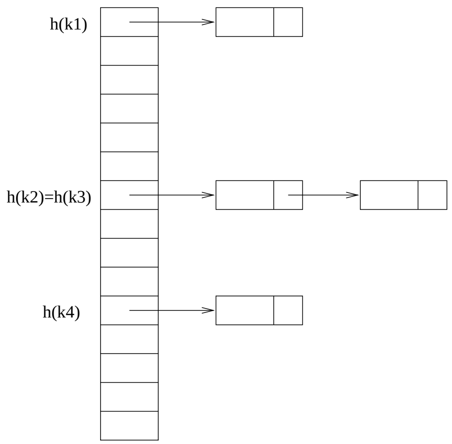
Prof. Dr. Franz Schweiggert — Sektion Angewandte Informationsverarbeitung — November 6, 1997
Ingo Melzer Assignment 4

Systemnahe Software I
Allgemeine Informatik III (WS 97/98)

Part 1 is due on November 18, 1997. Part 2 on November 25, 1997.

Hashing With Chaining

A lot of database applications only need the dictionary operations *insert, search,* and *delete*. Often hashing is used by those applications and their tables are therefore called hash tables. If you want to store an element with key k into your hash table of size m, you simple store it at *h(k)*. In other words, the hash function h is used to compute the slot from the key k, or h maps the universe U of keys into the slots of a hash table $T[0 \ldots m-1]$. The hash function we are going to use is $h(k) = k \bmod m$. The fly in the ointment of this beautiful idea is that two keys may hash to the same slot (h is usually not an injective function) – a collision. One of the easiest solutions is called "collision resolution by chaining". All elements that hash to the same slot are simply put into a linked list:



6 Hashing With Chaining (Part 1 — 10 points)

Start implementing such a hash table. You should provide to following functions: `insert(p,m,k)`, `printTable(p,m)` and `delete_all(p,m)`, where p is a pointer to a hash table of size m, and k the element to use. `print` should print your hash table, `delete_all` should delete the whole table and free memory allocated earlier. Insert the following elements into a hash table of size 9: 5, 28, 19, 15, 20, 33, 12, 17, 10.

7 Hashing With Chaining (Part 2 — 10 points)

Add the following functions to your solution of part 1: `delete_one(p,m,k)`, `search(p,m,k)`, `rehash(p,m1,m2)` and `utilization(p,m)`. The last function should print out the average length of a chain as well as the longest chain. `rehash` should transform a table of size *m1* into one of size *m2*. Make sure that no element gets lost and free unused memory.

Some Hints

Here are parts of a possible solution:

```
#define FALSE 0
#define TRUE 1
typedef int BOOL;

typedef struct node *nodePtr;
struct node {
   int key;
   /* Add your stuff here */
   nodePtr next;
} nodeList;

typedef struct {
   nodePtr p;
} hashtable;

/* Add some lines here */

void printTable(hashtable *p, int m) {
   /* Add some lines here */
}

void delete_all(hashtable *p, int m) {
   /* Add some lines here */
}

void insert(hashtable *p, int m, int k) {
   /* Add some lines here */
}

BOOL delete_one(hashtable *p, int m, int k) {
   /* Add some lines here */
}

/* And even more lines here :-) */
```