§3 A perfect hash table for integers

Listing 1: `perf_hash.c`

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define BLOCKSIZE 256
5
6  typedef int object_t;
7  typedef int key_t;
8
9  #define MAXP 46337  /* prime, and 46337*46337 < 2147483647 */
10
11 typedef struct { int          size;
12                   int     primary_a;
13                   int  *secondary_a;
14                   int  *secondary_s;
15                   int  *secondary_o;
16                   int         *keys;
17                   object_t    *objs; } perf_hash_t;
18
19 perf_hash_t *create_perf_hash(int size, int keys[],
20                                          object_t objs[])
21 {   perf_hash_t *tmp;
22     int *table1, *table2, *table3, *table4;
23     int i, j, k, collision, sq_bucket_sum, sq_sum_limit, a;
24     object_t *objects;
25     tmp = (perf_hash_t *) malloc( sizeof(perf_hash_t) );
26     table1 = (int *) malloc( size * sizeof(int) );
27     table2 = (int *) malloc( size * sizeof(int) );
28     table3 = (int *) malloc( size * sizeof(int) );
29     sq_sum_limit  =   5*size;
30     sq_bucket_sum = 100*size;
31     while(sq_bucket_sum > sq_sum_limit) /* find primary factor */
32       {  a = rand()%MAXP;
33        for(i=0; i<size; i++)
34          table1[i] = 0;
35        for(i=0; i<size; i++)
36          table1[ (((a*keys[i])%MAXP)% size) ] +=1;
37        sq_bucket_sum = 0;
38        for(i=0; i<size; i++)
39           sq_bucket_sum += table1[i]*table1[i];
40     }
41     /* compute secondary table sizes and their offset */
42     for(i=0; i< size; i++ )
43     {   table1[i] = 2*table1[i]*table1[i];
44         table2[i] = (i>0) ? table2[i-1] + table1[i-1] : 0;
45     }
46     table4 = (int *) malloc( 2*sq_bucket_sum * sizeof(int) );
```

```
47      for( i=0; i< 2*sq_bucket_sum; i++ )
48          table4[i] = MAXP; /* different from all keys */
49      collision = 1;
50      for( i=0; i< size; i++ )
51          table3[i] = rand()%MAXP; /* secondary hash factor */
52      while( collision )
53      {  collision = 0;
54         for( i=0; i< size; i++ )
55         { j = ((keys[i]*a)% MAXP) % size;
56           k = ((keys[i]*table3[j])% MAXP) % table1[j] + table2[j];
57           if( table4[k] == MAXP || table4[k] == keys[i] )
58               table4[k] = keys[i]; /* entry up to now empty */
59           else /* collision */
60           {  collision = 1;
61              table3[i] = 0; /* mark bucket as defect */
62           }
63         }
64         if( collision )
65         { for( i=0; i< size; i++)
66           { if( table3[i] == 0 ) /* defect bucket */
67             {  table3[i] = rand()%MAXP; /* choose new factor */
68                for( k= table2[i]; k< table2[i]+table1[i]; k++)
69                   table4[k] = MAXP; /* clear i-th secondary table */
70             }
71           }
72         }
73      } /* now the hash table is collision-free */
74      /* keys are in the right places, now put objects there */
75      objects =(object_t *)malloc(2*sq_bucket_sum*sizeof(object_t) );
76      for( i=0; i< size; i++ )
77      { j = ((keys[i]*a)% MAXP) % size;
78        k = ((keys[i]*table3[j])% MAXP) % table1[j] + table2[j];
79        objects[k] = objs[i];
80      }
81      tmp->size = size;
82      tmp->primary_a = a;          /* primary hash table factor */
83      tmp->secondary_a = table3; /* secondary hash table factors */
84      tmp->secondary_s = table1; /* secondary hash table sizes */
85      tmp->secondary_o = table2; /* secondary hash table offsets */
86      tmp->keys = table4; /* secondary hash tables */
87      tmp->objs = objects;
88      return( tmp );
89  }
90
91
92  object_t *find(perf_hash_t *ht, int query_key)
93  {  int i, j;
94     i = ((ht->primary_a*query_key)% MAXP)%ht->size;
95     if( ht->secondary_s[i] == 0 )
96       return( NULL ); /* secondary bucket empty */
97     else
```

```
98      {  j = ((ht->secondary_a[i]*query_key)% MAXP)%ht->secondary_s[i]
99             + ht->secondary_o[i];
100        if( ht->keys[j] == query_key )
101           return( (ht->objs)+j ); /* right key found */
102        else
103           return( NULL ); /* query_key does not exist. */
104     }
105  }
106
107
108
109  int main()
110  {
111     char nextop;
112     int keys[1000]; int objects[1000]; int size = 0; int i;
113     perf_hash_t *ht;
114     printf("Enter Keys (here we choose object 10*k for key k)\n");
115     while( (nextop = getchar())!= 'q' )
116     { if( nextop == 'i' )
117       { int inskey;
118         scanf(" %d", &inskey);
119         keys[size] = inskey;
120         objects[size] = 10*inskey;
121         size += 1;
122       }
123     }
124     printf("\nList of keys:\n");
125     for( i =0; i < size ; i++ )
126       printf(" %d", keys[i] );
127     printf("\n");
128     ht = create_perf_hash( size, keys, objects );
129     printf("created perfect hash table\n");
130     while( (nextop = getchar())!= 'q' )
131     {
132       if( nextop == 'f' )
133       { int findkey, *findobj;
134         scanf(" %d", &findkey);
135         printf(" looking for key %d\n", findkey);
136         findobj = find( ht, findkey);
137         if( findobj == NULL )
138           printf("  find failed, for key %d\n", findkey);
139         else
140           printf("  find successful, found object %d\n", *findobj);
141       }
142     }
143     return(0);
144  }
```