

no title

The Axiom Team

July 29, 2014

Abstract

Contents

```
/*
Copyright (c) 1991-2002, The Numerical Algorithms Group Ltd.
All rights reserved.
```

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of The Numerical Algorithms Group Ltd. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

```
*/
```

```
__ * __
```

```
#define _HASH_C
#include "debug.h"
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "hash.h"
```

```
#include "hash.h1"
#include "hallo.h1"
```

```
/* initialize a hash table */
```

```

void
hash_init(HashTable *table, int size, EqualFunction equal,
          HashcodeFunction hash_code)
{
    int i;

    table->table =
        (HashEntry **) malloc(size * sizeof(HashEntry *), "HashEntry");
    for (i = 0; i < size; i++)
        table->table[i] = NULL;
    table->size = size;
    table->equal = equal;
    table->hash_code = hash_code;
    table->num_entries = 0;
}

void
free_hash(HashTable *table, FreeFunction free_fun)
{
    if (table) {
        int i;

        for (i = 0; i < table->size; i++) {
            HashEntry *e, *next;

            for (e = table->table[i]; e != NULL;) {
                next = e->next;
                (*free_fun) (e->data);
                (*e).data=0;
                free(e);
                e = next;
            }
        }
        free(table->table);
    }
}

/* insert an entry into a hash table */

void
hash_insert(HashTable *table, char *data, char *key)
{
    HashEntry *entry = (HashEntry *) malloc(sizeof(HashEntry), "HashEntry");
    int code;

    entry->data = data;
    entry->key = key;
    code = (*table->hash_code) (key, table->size) % table->size;
#ifdef DEBUG

```

```

        fprintf(stderr, "Hash value = %d\n", code);
    #endif
    entry->next = table->table[code];
    table->table[code] = entry;
    table->num_entries++;
}

char *
hash_find(HashTable *table, char *key)
{
    HashEntry *entry;
    int code = table->hash_code(key, table->size) % table->size;

    for (entry = table->table[code]; entry != NULL; entry = entry->next)
        if ((*table->equal) (entry->key, key))
            return entry->data;
    return NULL;
}

char *
hash_replace(HashTable *table, char *data, char *key)
{
    HashEntry *entry;
    int code = table->hash_code(key, table->size) % table->size;

    for (entry = table->table[code]; entry != NULL; entry = entry->next)
        if ((*table->equal) (entry->key, key)) {
            entry->data = data;
            return entry->data;
        }
    return NULL;
}

void
hash_delete(HashTable *table, char *key)
{
    HashEntry **entry;
    int code = table->hash_code(key, table->size) % table->size;

    for (entry = &table->table[code]; *entry != NULL; entry = &((*entry)->next))
        if ((*table->equal) ((*entry)->key, key)) {
            *entry = (*entry)->next;
            table->num_entries--;
            return;
        }
}

void
hash_map(HashTable *table, MappableFunction func)
{

```

```

    int i;
    HashEntry *e;

    if (table == NULL)
        return;
    for (i = 0; i < table->size; i++)
        for (e = table->table[i]; e != NULL; e = e->next)
            (*func) (e->data);
}

HashEntry *
hash_copy_entry(HashEntry *e)
{
    HashEntry *ne;

    if (e == NULL)
        return e;
    ne = (HashEntry *) malloc(sizeof(HashEntry), "HashEntry");
    ne->data = e->data;
    ne->key = e->key;
    ne->next = hash_copy_entry(e->next);
    return ne;
}

/* copy a hash table */
HashTable *
hash_copy_table(HashTable *table)
{
    HashTable *nt = (HashTable *) malloc(sizeof(HashTable), "copy hash table");
    int i;

    nt->size = table->size;
    nt->num_entries = table->num_entries;
    nt->equal = table->equal;
    nt->hash_code = table->hash_code;
    nt->table = (HashEntry **) malloc(nt->size * sizeof(HashEntry *),
                                     "copy table");

    for (i = 0; i < table->size; i++)
        nt->table[i] = hash_copy_entry(table->table[i]);
    return nt;
}

/* hash code function for strings */
int
string_hash(char *s, int size)
{
    int c = 0;
    char *p = s;

```

```

        while (*p)
            c += *p++;
        return c % size;
    }

    /* test strings for equality */

    int
    string_equal(char *s1, char *s2)
    {
        return (strcmp(s1, s2) == 0);
    }

    /* make a fresh copy of the given string */
    char *
    alloc_string(char *str)
    {
        char * result;
        result = malloc(strlen(str)+1,"String");
        strcpy(result,str);
        return (result);
    }

```

References

- [1] nothing