/\* The MIT License

Copyright (c) 2008, 2009, 2011 by Attractive Chaos <attractor@live.co.uk>

Permission is hereby granted, free of charge, to any person obtaining

a copy of this software and associated documentation files (the

"Software"), to deal in the Software without restriction, including

without limitation the rights to use, copy, modify, merge, publish,

distribute, sublicense, and/or sell copies of the Software, and to

permit persons to whom the Software is furnished to do so, subject to

the following conditions:

The above copyright notice and this permission notice shall be

included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,

EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF

MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND

NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS

BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN

ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN

CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE

SOFTWARE.

\*/

/\*

An example:

#include "khash.h"

KHASH\_MAP\_INIT\_INT(32, char)

int main() {

int ret, is\_missing;

khiter\_t k;

khash\_t(32) \*h = kh\_init(32);

k = kh\_put(32, h, 5, &ret);

kh\_value(h, k) = 10;

k = kh\_get(32, h, 10);

is\_missing = (k == kh\_end(h));

k = kh\_get(32, h, 5);

kh\_del(32, h, k);

for (k = kh\_begin(h); k != kh\_end(h); ++k)

if (kh\_exist(h, k)) kh\_value(h, k) = 1;

kh\_destroy(32, h);

return 0;

}

\*/

/\*

2013-05-02 (0.2.8):

\* Use quadratic probing. When the capacity is power of 2, stepping function

i\*(i+1)/2 guarantees to traverse each bucket. It is better than double

hashing on cache performance and is more robust than linear probing.

In theory, double hashing should be more robust than quadratic probing.

However, my implementation is probably not for large hash tables, because

the second hash function is closely tied to the first hash function,

which reduce the effectiveness of double hashing.

Reference: http://research.cs.vt.edu/AVresearch/hashing/quadratic.php

2011-12-29 (0.2.7):

\* Minor code clean up; no actual effect.

2011-09-16 (0.2.6):

\* The capacity is a power of 2. This seems to dramatically improve the

speed for simple keys. Thank Zilong Tan for the suggestion. Reference:

- http://code.google.com/p/ulib/

- http://nothings.org/computer/judy/

\* Allow to optionally use linear probing which usually has better

performance for random input. Double hashing is still the default as it

is more robust to certain non-random input.

\* Added Wang's integer hash function (not used by default). This hash

function is more robust to certain non-random input.

2011-02-14 (0.2.5):

\* Allow to declare global functions.

2009-09-26 (0.2.4):

\* Improve portability

2008-09-19 (0.2.3):

\* Corrected the example

\* Improved interfaces

2008-09-11 (0.2.2):

\* Improved speed a little in kh\_put()

2008-09-10 (0.2.1):

\* Added kh\_clear()

\* Fixed a compiling error

2008-09-02 (0.2.0):

\* Changed to token concatenation which increases flexibility.

2008-08-31 (0.1.2):

\* Fixed a bug in kh\_get(), which has not been tested previously.

2008-08-31 (0.1.1):

\* Added destructor

\*/

#ifndef \_\_AC\_KHASH\_H

#define \_\_AC\_KHASH\_H

/\*!

@header

Generic hash table library.

\*/

#define AC\_VERSION\_KHASH\_H "0.2.8"

#include <stdlib.h>

#include <string.h>

#include <limits.h>

/\* compiler specific configuration \*/

#if UINT\_MAX == 0xffffffffu

typedef unsigned int khint32\_t;

#elif ULONG\_MAX == 0xffffffffu

typedef unsigned long khint32\_t;

#endif

#if ULONG\_MAX == ULLONG\_MAX

typedef unsigned long khint64\_t;

#else

typedef uint64\_t khint64\_t;

#endif

#ifndef kh\_inline

#ifdef \_MSC\_VER

#define kh\_inline \_\_inline

#else

#define kh\_inline inline

#endif

#endif /\* kh\_inline \*/

#ifndef klib\_unused

#if (defined \_\_clang\_\_ && \_\_clang\_major\_\_ >= 3) || (defined \_\_GNUC\_\_ && \_\_GNUC\_\_ >= 3)

#define klib\_unused \_\_attribute\_\_ ((\_\_unused\_\_))

#else

#define klib\_unused

#endif

#endif /\* klib\_unused \*/

typedef khint32\_t khint\_t;

typedef khint\_t khiter\_t;

#define \_\_ac\_isempty(flag, i) ((flag[i>>4]>>((i&0xfU)<<1))&2)

#define \_\_ac\_isdel(flag, i) ((flag[i>>4]>>((i&0xfU)<<1))&1)

#define \_\_ac\_iseither(flag, i) ((flag[i>>4]>>((i&0xfU)<<1))&3)

#define \_\_ac\_set\_isdel\_false(flag, i) (flag[i>>4]&=~(1ul<<((i&0xfU)<<1)))

#define \_\_ac\_set\_isempty\_false(flag, i) (flag[i>>4]&=~(2ul<<((i&0xfU)<<1)))

#define \_\_ac\_set\_isboth\_false(flag, i) (flag[i>>4]&=~(3ul<<((i&0xfU)<<1)))

#define \_\_ac\_set\_isdel\_true(flag, i) (flag[i>>4]|=1ul<<((i&0xfU)<<1))

#define \_\_ac\_fsize(m) ((m) < 16? 1 : (m)>>4)

#ifndef kroundup32

#define kroundup32(x) (--(x), (x)|=(x)>>1, (x)|=(x)>>2, (x)|=(x)>>4, (x)|=(x)>>8, (x)|=(x)>>16, ++(x))

#endif

#ifndef kcalloc

#define kcalloc(N,Z) calloc(N,Z)

#endif

#ifndef kmalloc

#define kmalloc(Z) malloc(Z)

#endif

#ifndef krealloc

#define krealloc(P,Z) realloc(P,Z)

#endif

#ifndef kfree

#define kfree(P) free(P)

#endif

static const double \_\_ac\_HASH\_UPPER = 0.77;

#define \_\_KHASH\_TYPE(name, khkey\_t, khval\_t) \

typedef struct kh\_##name##\_s { \

khint\_t n\_buckets, size, n\_occupied, upper\_bound; \

khint32\_t \*flags; \

khkey\_t \*keys; \

khval\_t \*vals; \

} kh\_##name##\_t;

#define \_\_KHASH\_PROTOTYPES(name, khkey\_t, khval\_t) \

extern kh\_##name##\_t \*kh\_init\_##name(void); \

extern void kh\_destroy\_##name(kh\_##name##\_t \*h); \

extern void kh\_clear\_##name(kh\_##name##\_t \*h); \

extern khint\_t kh\_get\_##name(const kh\_##name##\_t \*h, khkey\_t key); \

extern int kh\_resize\_##name(kh\_##name##\_t \*h, khint\_t new\_n\_buckets); \

extern khint\_t kh\_put\_##name(kh\_##name##\_t \*h, khkey\_t key, int \*ret); \

extern void kh\_del\_##name(kh\_##name##\_t \*h, khint\_t x);

#define \_\_KHASH\_IMPL(name, SCOPE, khkey\_t, khval\_t, kh\_is\_map, \_\_hash\_func, \_\_hash\_equal) \

SCOPE kh\_##name##\_t \*kh\_init\_##name(void) { \

return (kh\_##name##\_t\*)kcalloc(1, sizeof(kh\_##name##\_t)); \

} \

SCOPE void kh\_destroy\_##name(kh\_##name##\_t \*h) \

{ \

if (h) { \

kfree((void \*)h->keys); kfree(h->flags); \

kfree((void \*)h->vals); \

kfree(h); \

} \

} \

SCOPE void kh\_clear\_##name(kh\_##name##\_t \*h) \

{ \

if (h && h->flags) { \

memset(h->flags, 0xaa, \_\_ac\_fsize(h->n\_buckets) \* sizeof(khint32\_t)); \

h->size = h->n\_occupied = 0; \

} \

} \

SCOPE khint\_t kh\_get\_##name(const kh\_##name##\_t \*h, khkey\_t key) \

{ \

if (h->n\_buckets) { \

khint\_t k, i, last, mask, step = 0; \

mask = h->n\_buckets - 1; \

k = \_\_hash\_func(key); i = k & mask; \

last = i; \

while (!\_\_ac\_isempty(h->flags, i) && (\_\_ac\_isdel(h->flags, i) || !\_\_hash\_equal(h->keys[i], key))) { \

i = (i + (++step)) & mask; \

if (i == last) return h->n\_buckets; \

} \

return \_\_ac\_iseither(h->flags, i)? h->n\_buckets : i; \

} else return 0; \

} \

SCOPE int kh\_resize\_##name(kh\_##name##\_t \*h, khint\_t new\_n\_buckets) \

{ /\* This function uses 0.25\*n\_buckets bytes of working space instead of [sizeof(key\_t+val\_t)+.25]\*n\_buckets. \*/ \

khint32\_t \*new\_flags = 0; \

khint\_t j = 1; \

{ \

kroundup32(new\_n\_buckets); \

if (new\_n\_buckets < 4) new\_n\_buckets = 4; \

if (h->size >= (khint\_t)(new\_n\_buckets \* \_\_ac\_HASH\_UPPER + 0.5)) j = 0; /\* requested size is too small \*/ \

else { /\* hash table size to be changed (shrink or expand); rehash \*/ \

new\_flags = (khint32\_t\*)kmalloc(\_\_ac\_fsize(new\_n\_buckets) \* sizeof(khint32\_t)); \

if (!new\_flags) return -1; \

memset(new\_flags, 0xaa, \_\_ac\_fsize(new\_n\_buckets) \* sizeof(khint32\_t)); \

if (h->n\_buckets < new\_n\_buckets) { /\* expand \*/ \

khkey\_t \*new\_keys = (khkey\_t\*)krealloc((void \*)h->keys, new\_n\_buckets \* sizeof(khkey\_t)); \

if (!new\_keys) { kfree(new\_flags); return -1; } \

h->keys = new\_keys; \

if (kh\_is\_map) { \

khval\_t \*new\_vals = (khval\_t\*)krealloc((void \*)h->vals, new\_n\_buckets \* sizeof(khval\_t)); \

if (!new\_vals) { kfree(new\_flags); return -1; } \

h->vals = new\_vals; \

} \

} /\* otherwise shrink \*/ \

} \

} \

if (j) { /\* rehashing is needed \*/ \

for (j = 0; j != h->n\_buckets; ++j) { \

if (\_\_ac\_iseither(h->flags, j) == 0) { \

khkey\_t key = h->keys[j]; \

khval\_t val; \

khint\_t new\_mask; \

new\_mask = new\_n\_buckets - 1; \

if (kh\_is\_map) val = h->vals[j]; \

\_\_ac\_set\_isdel\_true(h->flags, j); \

while (1) { /\* kick-out process; sort of like in Cuckoo hashing \*/ \

khint\_t k, i, step = 0; \

k = \_\_hash\_func(key); \

i = k & new\_mask; \

while (!\_\_ac\_isempty(new\_flags, i)) i = (i + (++step)) & new\_mask; \

\_\_ac\_set\_isempty\_false(new\_flags, i); \

if (i < h->n\_buckets && \_\_ac\_iseither(h->flags, i) == 0) { /\* kick out the existing element \*/ \

{ khkey\_t tmp = h->keys[i]; h->keys[i] = key; key = tmp; } \

if (kh\_is\_map) { khval\_t tmp = h->vals[i]; h->vals[i] = val; val = tmp; } \

\_\_ac\_set\_isdel\_true(h->flags, i); /\* mark it as deleted in the old hash table \*/ \

} else { /\* write the element and jump out of the loop \*/ \

h->keys[i] = key; \

if (kh\_is\_map) h->vals[i] = val; \

break; \

} \

} \

} \

} \

if (h->n\_buckets > new\_n\_buckets) { /\* shrink the hash table \*/ \

h->keys = (khkey\_t\*)krealloc((void \*)h->keys, new\_n\_buckets \* sizeof(khkey\_t)); \

if (kh\_is\_map) h->vals = (khval\_t\*)krealloc((void \*)h->vals, new\_n\_buckets \* sizeof(khval\_t)); \

} \

kfree(h->flags); /\* free the working space \*/ \

h->flags = new\_flags; \

h->n\_buckets = new\_n\_buckets; \

h->n\_occupied = h->size; \

h->upper\_bound = (khint\_t)(h->n\_buckets \* \_\_ac\_HASH\_UPPER + 0.5); \

} \

return 0; \

} \

SCOPE khint\_t kh\_put\_##name(kh\_##name##\_t \*h, khkey\_t key, int \*ret) \

{ \

khint\_t x; \

if (h->n\_occupied >= h->upper\_bound) { /\* update the hash table \*/ \

if (h->n\_buckets > (h->size<<1)) { \

if (kh\_resize\_##name(h, h->n\_buckets - 1) < 0) { /\* clear "deleted" elements \*/ \

\*ret = -1; return h->n\_buckets; \

} \

} else if (kh\_resize\_##name(h, h->n\_buckets + 1) < 0) { /\* expand the hash table \*/ \

\*ret = -1; return h->n\_buckets; \

} \

} /\* TODO: to implement automatically shrinking; resize() already support shrinking \*/ \

{ \

khint\_t k, i, site, last, mask = h->n\_buckets - 1, step = 0; \

x = site = h->n\_buckets; k = \_\_hash\_func(key); i = k & mask; \

if (\_\_ac\_isempty(h->flags, i)) x = i; /\* for speed up \*/ \

else { \

last = i; \

while (!\_\_ac\_isempty(h->flags, i) && (\_\_ac\_isdel(h->flags, i) || !\_\_hash\_equal(h->keys[i], key))) { \

if (\_\_ac\_isdel(h->flags, i)) site = i; \

i = (i + (++step)) & mask; \

if (i == last) { x = site; break; } \

} \

if (x == h->n\_buckets) { \

if (\_\_ac\_isempty(h->flags, i) && site != h->n\_buckets) x = site; \

else x = i; \

} \

} \

} \

if (\_\_ac\_isempty(h->flags, x)) { /\* not present at all \*/ \

h->keys[x] = key; \

\_\_ac\_set\_isboth\_false(h->flags, x); \

++h->size; ++h->n\_occupied; \

\*ret = 1; \

} else if (\_\_ac\_isdel(h->flags, x)) { /\* deleted \*/ \

h->keys[x] = key; \

\_\_ac\_set\_isboth\_false(h->flags, x); \

++h->size; \

\*ret = 2; \

} else \*ret = 0; /\* Don't touch h->keys[x] if present and not deleted \*/ \

return x; \

} \

SCOPE void kh\_del\_##name(kh\_##name##\_t \*h, khint\_t x) \

{ \

if (x != h->n\_buckets && !\_\_ac\_iseither(h->flags, x)) { \

\_\_ac\_set\_isdel\_true(h->flags, x); \

--h->size; \

} \

}

#define KHASH\_DECLARE(name, khkey\_t, khval\_t) \

\_\_KHASH\_TYPE(name, khkey\_t, khval\_t) \

\_\_KHASH\_PROTOTYPES(name, khkey\_t, khval\_t)

#define KHASH\_INIT2(name, SCOPE, khkey\_t, khval\_t, kh\_is\_map, \_\_hash\_func, \_\_hash\_equal) \

\_\_KHASH\_TYPE(name, khkey\_t, khval\_t) \

\_\_KHASH\_IMPL(name, SCOPE, khkey\_t, khval\_t, kh\_is\_map, \_\_hash\_func, \_\_hash\_equal)

#define KHASH\_INIT(name, khkey\_t, khval\_t, kh\_is\_map, \_\_hash\_func, \_\_hash\_equal) \

KHASH\_INIT2(name, static kh\_inline klib\_unused, khkey\_t, khval\_t, kh\_is\_map, \_\_hash\_func, \_\_hash\_equal)

/\* --- BEGIN OF HASH FUNCTIONS --- \*/

/\*! @function

@abstract Integer hash function

@param key The integer [khint32\_t]

@return The hash value [khint\_t]

\*/

#define kh\_int\_hash\_func(key) (khint32\_t)(key)

/\*! @function

@abstract Integer comparison function

\*/

#define kh\_int\_hash\_equal(a, b) ((a) == (b))

/\*! @function

@abstract 64-bit integer hash function

@param key The integer [khint64\_t]

@return The hash value [khint\_t]

\*/

#define kh\_int64\_hash\_func(key) (khint32\_t)((key)>>33^(key)^(key)<<11)

/\*! @function

@abstract 64-bit integer comparison function

\*/

#define kh\_int64\_hash\_equal(a, b) ((a) == (b))

/\*! @function

@abstract const char\* hash function

@param s Pointer to a null terminated string

@return The hash value

\*/

static kh\_inline khint\_t \_\_ac\_X31\_hash\_string(const char \*s)

{

khint\_t h = (khint\_t)\*s;

if (h) for (++s ; \*s; ++s) h = (h << 5) - h + (khint\_t)\*s;

return h;

}

/\*! @function

@abstract Another interface to const char\* hash function

@param key Pointer to a null terminated string [const char\*]

@return The hash value [khint\_t]

\*/

#define kh\_str\_hash\_func(key) \_\_ac\_X31\_hash\_string(key)

/\*! @function

@abstract Const char\* comparison function

\*/

#define kh\_str\_hash\_equal(a, b) (strcmp(a, b) == 0)

static kh\_inline khint\_t \_\_ac\_Wang\_hash(khint\_t key)

{

key += ~(key << 15);

key ^= (key >> 10);

key += (key << 3);

key ^= (key >> 6);

key += ~(key << 11);

key ^= (key >> 16);

return key;

}

#define kh\_int\_hash\_func2(key) \_\_ac\_Wang\_hash((khint\_t)key)

/\* --- END OF HASH FUNCTIONS --- \*/

/\* Other convenient macros... \*/

/\*!

@abstract Type of the hash table.

@param name Name of the hash table [symbol]

\*/

#define khash\_t(name) kh\_##name##\_t

/\*! @function

@abstract Initiate a hash table.

@param name Name of the hash table [symbol]

@return Pointer to the hash table [khash\_t(name)\*]

\*/

#define kh\_init(name) kh\_init\_##name()

/\*! @function

@abstract Destroy a hash table.

@param name Name of the hash table [symbol]

@param h Pointer to the hash table [khash\_t(name)\*]

\*/

#define kh\_destroy(name, h) kh\_destroy\_##name(h)

/\*! @function

@abstract Reset a hash table without deallocating memory.

@param name Name of the hash table [symbol]

@param h Pointer to the hash table [khash\_t(name)\*]

\*/

#define kh\_clear(name, h) kh\_clear\_##name(h)

/\*! @function

@abstract Resize a hash table.

@param name Name of the hash table [symbol]

@param h Pointer to the hash table [khash\_t(name)\*]

@param s New size [khint\_t]

\*/

#define kh\_resize(name, h, s) kh\_resize\_##name(h, s)

/\*! @function

@abstract Insert a key to the hash table.

@param name Name of the hash table [symbol]

@param h Pointer to the hash table [khash\_t(name)\*]

@param k Key [type of keys]

@param r Extra return code: -1 if the operation failed;

0 if the key is present in the hash table;

1 if the bucket is empty (never used); 2 if the element in

the bucket has been deleted [int\*]

@return Iterator to the inserted element [khint\_t]

\*/

#define kh\_put(name, h, k, r) kh\_put\_##name(h, k, r)

/\*! @function

@abstract Retrieve a key from the hash table.

@param name Name of the hash table [symbol]

@param h Pointer to the hash table [khash\_t(name)\*]

@param k Key [type of keys]

@return Iterator to the found element, or kh\_end(h) if the element is absent [khint\_t]

\*/

#define kh\_get(name, h, k) kh\_get\_##name(h, k)

/\*! @function

@abstract Remove a key from the hash table.

@param name Name of the hash table [symbol]

@param h Pointer to the hash table [khash\_t(name)\*]

@param k Iterator to the element to be deleted [khint\_t]

\*/

#define kh\_del(name, h, k) kh\_del\_##name(h, k)

/\*! @function

@abstract Test whether a bucket contains data.

@param h Pointer to the hash table [khash\_t(name)\*]

@param x Iterator to the bucket [khint\_t]

@return 1 if containing data; 0 otherwise [int]

\*/

#define kh\_exist(h, x) (!\_\_ac\_iseither((h)->flags, (x)))

/\*! @function

@abstract Get key given an iterator

@param h Pointer to the hash table [khash\_t(name)\*]

@param x Iterator to the bucket [khint\_t]

@return Key [type of keys]

\*/

#define kh\_key(h, x) ((h)->keys[x])

/\*! @function

@abstract Get value given an iterator

@param h Pointer to the hash table [khash\_t(name)\*]

@param x Iterator to the bucket [khint\_t]

@return Value [type of values]

@discussion For hash sets, calling this results in segfault.

\*/

#define kh\_val(h, x) ((h)->vals[x])

/\*! @function

@abstract Alias of kh\_val()

\*/

#define kh\_value(h, x) ((h)->vals[x])

/\*! @function

@abstract Get the start iterator

@param h Pointer to the hash table [khash\_t(name)\*]

@return The start iterator [khint\_t]

\*/

#define kh\_begin(h) (khint\_t)(0)

/\*! @function

@abstract Get the end iterator

@param h Pointer to the hash table [khash\_t(name)\*]

@return The end iterator [khint\_t]

\*/

#define kh\_end(h) ((h)->n\_buckets)

/\*! @function

@abstract Get the number of elements in the hash table

@param h Pointer to the hash table [khash\_t(name)\*]

@return Number of elements in the hash table [khint\_t]

\*/

#define kh\_size(h) ((h)->size)

/\*! @function

@abstract Get the number of buckets in the hash table

@param h Pointer to the hash table [khash\_t(name)\*]

@return Number of buckets in the hash table [khint\_t]

\*/

#define kh\_n\_buckets(h) ((h)->n\_buckets)

/\*! @function

@abstract Iterate over the entries in the hash table

@param h Pointer to the hash table [khash\_t(name)\*]

@param kvar Variable to which key will be assigned

@param vvar Variable to which value will be assigned

@param code Block of code to execute

\*/

#define kh\_foreach(h, kvar, vvar, code) { khint\_t \_\_i; \

for (\_\_i = kh\_begin(h); \_\_i != kh\_end(h); ++\_\_i) { \

if (!kh\_exist(h,\_\_i)) continue; \

(kvar) = kh\_key(h,\_\_i); \

(vvar) = kh\_val(h,\_\_i); \

code; \

} }

/\*! @function

@abstract Iterate over the values in the hash table

@param h Pointer to the hash table [khash\_t(name)\*]

@param vvar Variable to which value will be assigned

@param code Block of code to execute

\*/

#define kh\_foreach\_value(h, vvar, code) { khint\_t \_\_i; \

for (\_\_i = kh\_begin(h); \_\_i != kh\_end(h); ++\_\_i) { \

if (!kh\_exist(h,\_\_i)) continue; \

(vvar) = kh\_val(h,\_\_i); \

code; \

} }

/\* More conenient interfaces \*/

/\*! @function

@abstract Instantiate a hash set containing integer keys

@param name Name of the hash table [symbol]

\*/

#define KHASH\_SET\_INIT\_INT(name) \

KHASH\_INIT(name, khint32\_t, char, 0, kh\_int\_hash\_func, kh\_int\_hash\_equal)

/\*! @function

@abstract Instantiate a hash map containing integer keys

@param name Name of the hash table [symbol]

@param khval\_t Type of values [type]

\*/

#define KHASH\_MAP\_INIT\_INT(name, khval\_t) \

KHASH\_INIT(name, khint32\_t, khval\_t, 1, kh\_int\_hash\_func, kh\_int\_hash\_equal)

/\*! @function

@abstract Instantiate a hash map containing 64-bit integer keys

@param name Name of the hash table [symbol]

\*/

#define KHASH\_SET\_INIT\_INT64(name) \

KHASH\_INIT(name, khint64\_t, char, 0, kh\_int64\_hash\_func, kh\_int64\_hash\_equal)

/\*! @function

@abstract Instantiate a hash map containing 64-bit integer keys

@param name Name of the hash table [symbol]

@param khval\_t Type of values [type]

\*/

#define KHASH\_MAP\_INIT\_INT64(name, khval\_t) \

KHASH\_INIT(name, khint64\_t, khval\_t, 1, kh\_int64\_hash\_func, kh\_int64\_hash\_equal)

typedef const char \*kh\_cstr\_t;

/\*! @function

@abstract Instantiate a hash map containing const char\* keys

@param name Name of the hash table [symbol]

\*/

#define KHASH\_SET\_INIT\_STR(name) \

KHASH\_INIT(name, kh\_cstr\_t, char, 0, kh\_str\_hash\_func, kh\_str\_hash\_equal)

/\*! @function

@abstract Instantiate a hash map containing const char\* keys

@param name Name of the hash table [symbol]

@param khval\_t Type of values [type]

\*/

#define KHASH\_MAP\_INIT\_STR(name, khval\_t) \

KHASH\_INIT(name, kh\_cstr\_t, khval\_t, 1, kh\_str\_hash\_func, kh\_str\_hash\_equal)

#endif /\* \_\_AC\_KHASH\_H \*/