kaldi-utils.h

Go to the documentation of this file.

```
// base/kaldi-utils.h

// Copyright 2009-2011  Ondrej Glembek;  Microsoft Corporation;
//                      Saarland University;  Karel Vesely;  Yanmin Qian

// See ../../COPYING for clarification regarding multiple authors
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//  http://www.apache.org/licenses/LICENSE-2.0
//
// THIS CODE IS PROVIDED *AS IS* BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
// KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED
// WARRANTIES OR CONDITIONS OF TITLE, FITNESS FOR A PARTICULAR PURPOSE,
// MERCHANTABLITY OR NON-INFRINGEMENT.
// See the Apache 2 License for the specific language governing permissions and
// limitations under the License.

#ifndef KALDI_BASE_KALDI_UTILS_H_
#define KALDI_BASE_KALDI_UTILS_H_ 1

#include <limits>
#include <string>

#if defined(_MSC_VER)
# define WIN32_LEAN_AND_MEAN
# define NOMINMAX
# include <windows.h>
#endif

#if defined(_MSC_VER)
#pragma warning(disable: 4244 4056 4305 4800 4267 4996 4756 4661)
#if _MSC_VER < 1400
#define __restrict__
#else
#define __restrict__ __restrict
#endif
#endif

#ifdef HAVE_POSIX_MEMALIGN
#  define KALDI_MEMALIGN(align, size, pp_orig) \
     (!posix_memalign(pp_orig, align, size) ? *(pp_orig) : NULL)
#  define KALDI_MEMALIGN_FREE(x) free(x)
#elif defined(HAVE_MEMALIGN)
  /* Some systems have memalign() but no declaration for it */
  void * memalign(size_t align, size_t size);
#  define KALDI_MEMALIGN(align, size, pp_orig) \
     (*(pp_orig) = memalign(align, size))
#  define KALDI_MEMALIGN_FREE(x) free(x)
#elif defined(_MSC_VER)
#  define KALDI_MEMALIGN(align, size, pp_orig) \
  (*(pp_orig) = _aligned_malloc(size, align))
#  define KALDI_MEMALIGN_FREE(x) _aligned_free(x)
#else
#error Manual memory alignment is no longer supported
#endif

#ifdef __ICC
#pragma warning(disable: 383)  // ICPC remark we don't want.
#pragma warning(disable: 810)  // ICPC remark we don't want.
#pragma warning(disable: 981)  // ICPC remark we don't want.
#pragma warning(disable: 1418)  // ICPC remark we don't want.
#pragma warning(disable: 444)  // ICPC remark we don't want.
#pragma warning(disable: 869)  // ICPC remark we don't want.
#pragma warning(disable: 1287)  // ICPC remark we don't want.
#pragma warning(disable: 279)  // ICPC remark we don't want.
```

```
#pragma warning(disable: 981)  // ICPC remark we don't want.
#endif


namespace kaldi {


// CharToString prints the character in a human-readable form, for debugging.
std::string CharToString(const char &c);


inline int MachineIsLittleEndian() {
  int check = 1;
  return (*reinterpret_cast<char*>(&check) != 0);
}

// This function kaldi::Sleep() provides a portable way to sleep for a possibly
 fractional
// number of seconds.  On Windows it's only accurate to microseconds.
void Sleep(float seconds);

}

#define KALDI_SWAP8(a) { \
  int t = ((char*)&a)[0]; ((char*)&a)[0]=((char*)&a)[7]; ((char*)&a)[7]=t;\
      t = ((char*)&a)[1]; ((char*)&a)[1]=((char*)&a)[6]; ((char*)&a)[6]=t;\
      t = ((char*)&a)[2]; ((char*)&a)[2]=((char*)&a)[5]; ((char*)&a)[5]=t;\
      t = ((char*)&a)[3]; ((char*)&a)[3]=((char*)&a)[4]; ((char*)&a)[4]=t;}
#define KALDI_SWAP4(a) { \
  int t = ((char*)&a)[0]; ((char*)&a)[0]=((char*)&a)[3]; ((char*)&a)[3]=t;\
      t = ((char*)&a)[1]; ((char*)&a)[1]=((char*)&a)[2]; ((char*)&a)[2]=t;}
#define KALDI_SWAP2(a) { \
  int t = ((char*)&a)[0]; ((char*)&a)[0]=((char*)&a)[1]; ((char*)&a)[1]=t;}


// Makes copy constructor and operator= private.  Same as in compat.h of OpenFst
// toolkit.
#define KALDI_DISALLOW_COPY_AND_ASSIGN(type)    \
  type(const type&);                  \
  void operator = (const type&)

template<bool B> class KaldiCompileTimeAssert { };
template<> class KaldiCompileTimeAssert<true> {
 public:
  static inline void Check() { }
};

#define KALDI_COMPILE_TIME_ASSERT(b) KaldiCompileTimeAssert<(b)>::Check()

#define KALDI_ASSERT_IS_INTEGER_TYPE(I) \
  KaldiCompileTimeAssert<std::numeric_limits<I>::is_specialized \
                 && std::numeric_limits<I>::is_integer>::Check()

#define KALDI_ASSERT_IS_FLOATING_TYPE(F) \
  KaldiCompileTimeAssert<std::numeric_limits<F>::is_specialized \
                && !std::numeric_limits<F>::is_integer>::Check()

#ifdef _MSC_VER
#include <stdio.h>
#define unlink _unlink
#else
#include <unistd.h>
#endif


#ifdef _MSC_VER
#define KALDI_STRCASECMP _stricmp
#else
#define KALDI_STRCASECMP strcasecmp
#endif
#ifdef _MSC_VER
#  define KALDI_STRTOLL(cur_cstr, end_cstr) _strtoi64(cur_cstr, end_cstr, 10);
#else
#  define KALDI_STRTOLL(cur_cstr, end_cstr) strtoll(cur_cstr, end_cstr, 10);
#endif
```

```

#define KALDI_STRTOD(cur_cstr, end_cstr) strtod(cur_cstr, end_cstr)

#ifdef _MSC_VER
#  define KALDI_STRTOF(cur_cstr, end_cstr) \
    static_cast<float>(strtod(cur_cstr, end_cstr));
#else
#  define KALDI_STRTOF(cur_cstr, end_cstr) strtof(cur_cstr, end_cstr);
#endif

#endif  // KALDI_BASE_KALDI_UTILS_H_

```