#ifdef \_\_cplusplus

# error "A C++ compiler has been selected for C."

#endif

#if defined(\_\_18CXX)

# define ID\_VOID\_MAIN

#endif

#if defined(\_\_CLASSIC\_C\_\_)

/\* cv-qualifiers did not exist in K&R C \*/

# define const

# define volatile

#endif

#if !defined(\_\_has\_include)

/\* If the compiler does not have \_\_has\_include, pretend the answer is

always no. \*/

# define \_\_has\_include(x) 0

#endif

/\* Version number components: V=Version, R=Revision, P=Patch

Version date components: YYYY=Year, MM=Month, DD=Day \*/

#if defined(\_\_INTEL\_COMPILER) || defined(\_\_ICC)

# define COMPILER\_ID "Intel"

# if defined(\_MSC\_VER)

# define SIMULATE\_ID "MSVC"

# endif

# if defined(\_\_GNUC\_\_)

# define SIMULATE\_ID "GNU"

# endif

/\* \_\_INTEL\_COMPILER = VRP prior to 2021, and then VVVV for 2021 and later,

except that a few beta releases use the old format with V=2021. \*/

# if \_\_INTEL\_COMPILER < 2021 || \_\_INTEL\_COMPILER == 202110 || \_\_INTEL\_COMPILER == 202111

# define COMPILER\_VERSION\_MAJOR DEC(\_\_INTEL\_COMPILER/100)

# define COMPILER\_VERSION\_MINOR DEC(\_\_INTEL\_COMPILER/10 % 10)

# if defined(\_\_INTEL\_COMPILER\_UPDATE)

# define COMPILER\_VERSION\_PATCH DEC(\_\_INTEL\_COMPILER\_UPDATE)

# else

# define COMPILER\_VERSION\_PATCH DEC(\_\_INTEL\_COMPILER % 10)

# endif

# else

# define COMPILER\_VERSION\_MAJOR DEC(\_\_INTEL\_COMPILER)

# define COMPILER\_VERSION\_MINOR DEC(\_\_INTEL\_COMPILER\_UPDATE)

/\* The third version component from --version is an update index,

but no macro is provided for it. \*/

# define COMPILER\_VERSION\_PATCH DEC(0)

# endif

# if defined(\_\_INTEL\_COMPILER\_BUILD\_DATE)

/\* \_\_INTEL\_COMPILER\_BUILD\_DATE = YYYYMMDD \*/

# define COMPILER\_VERSION\_TWEAK DEC(\_\_INTEL\_COMPILER\_BUILD\_DATE)

# endif

# if defined(\_MSC\_VER)

/\* \_MSC\_VER = VVRR \*/

# define SIMULATE\_VERSION\_MAJOR DEC(\_MSC\_VER / 100)

# define SIMULATE\_VERSION\_MINOR DEC(\_MSC\_VER % 100)

# endif

# if defined(\_\_GNUC\_\_)

# define SIMULATE\_VERSION\_MAJOR DEC(\_\_GNUC\_\_)

# elif defined(\_\_GNUG\_\_)

# define SIMULATE\_VERSION\_MAJOR DEC(\_\_GNUG\_\_)

# endif

# if defined(\_\_GNUC\_MINOR\_\_)

# define SIMULATE\_VERSION\_MINOR DEC(\_\_GNUC\_MINOR\_\_)

# endif

# if defined(\_\_GNUC\_PATCHLEVEL\_\_)

# define SIMULATE\_VERSION\_PATCH DEC(\_\_GNUC\_PATCHLEVEL\_\_)

# endif

#elif (defined(\_\_clang\_\_) && defined(\_\_INTEL\_CLANG\_COMPILER)) || defined(\_\_INTEL\_LLVM\_COMPILER)

# define COMPILER\_ID "IntelLLVM"

#if defined(\_MSC\_VER)

# define SIMULATE\_ID "MSVC"

#endif

#if defined(\_\_GNUC\_\_)

# define SIMULATE\_ID "GNU"

#endif

/\* \_\_INTEL\_LLVM\_COMPILER = VVVVRP prior to 2021.2.0, VVVVRRPP for 2021.2.0 and

\* later. Look for 6 digit vs. 8 digit version number to decide encoding.

\* VVVV is no smaller than the current year when a version is released.

\*/

#if \_\_INTEL\_LLVM\_COMPILER < 1000000L

# define COMPILER\_VERSION\_MAJOR DEC(\_\_INTEL\_LLVM\_COMPILER/100)

# define COMPILER\_VERSION\_MINOR DEC(\_\_INTEL\_LLVM\_COMPILER/10 % 10)

# define COMPILER\_VERSION\_PATCH DEC(\_\_INTEL\_LLVM\_COMPILER % 10)

#else

# define COMPILER\_VERSION\_MAJOR DEC(\_\_INTEL\_LLVM\_COMPILER/10000)

# define COMPILER\_VERSION\_MINOR DEC(\_\_INTEL\_LLVM\_COMPILER/100 % 100)

# define COMPILER\_VERSION\_PATCH DEC(\_\_INTEL\_LLVM\_COMPILER % 100)

#endif

#if defined(\_MSC\_VER)

/\* \_MSC\_VER = VVRR \*/

# define SIMULATE\_VERSION\_MAJOR DEC(\_MSC\_VER / 100)

# define SIMULATE\_VERSION\_MINOR DEC(\_MSC\_VER % 100)

#endif

#if defined(\_\_GNUC\_\_)

# define SIMULATE\_VERSION\_MAJOR DEC(\_\_GNUC\_\_)

#elif defined(\_\_GNUG\_\_)

# define SIMULATE\_VERSION\_MAJOR DEC(\_\_GNUG\_\_)

#endif

#if defined(\_\_GNUC\_MINOR\_\_)

# define SIMULATE\_VERSION\_MINOR DEC(\_\_GNUC\_MINOR\_\_)

#endif

#if defined(\_\_GNUC\_PATCHLEVEL\_\_)

# define SIMULATE\_VERSION\_PATCH DEC(\_\_GNUC\_PATCHLEVEL\_\_)

#endif

#elif defined(\_\_PATHCC\_\_)

# define COMPILER\_ID "PathScale"

# define COMPILER\_VERSION\_MAJOR DEC(\_\_PATHCC\_\_)

# define COMPILER\_VERSION\_MINOR DEC(\_\_PATHCC\_MINOR\_\_)

# if defined(\_\_PATHCC\_PATCHLEVEL\_\_)

# define COMPILER\_VERSION\_PATCH DEC(\_\_PATHCC\_PATCHLEVEL\_\_)

# endif

#elif defined(\_\_BORLANDC\_\_) && defined(\_\_CODEGEARC\_VERSION\_\_)

# define COMPILER\_ID "Embarcadero"

# define COMPILER\_VERSION\_MAJOR HEX(\_\_CODEGEARC\_VERSION\_\_>>24 & 0x00FF)

# define COMPILER\_VERSION\_MINOR HEX(\_\_CODEGEARC\_VERSION\_\_>>16 & 0x00FF)

# define COMPILER\_VERSION\_PATCH DEC(\_\_CODEGEARC\_VERSION\_\_ & 0xFFFF)

#elif defined(\_\_BORLANDC\_\_)

# define COMPILER\_ID "Borland"

/\* \_\_BORLANDC\_\_ = 0xVRR \*/

# define COMPILER\_VERSION\_MAJOR HEX(\_\_BORLANDC\_\_>>8)

# define COMPILER\_VERSION\_MINOR HEX(\_\_BORLANDC\_\_ & 0xFF)

#elif defined(\_\_WATCOMC\_\_) && \_\_WATCOMC\_\_ < 1200

# define COMPILER\_ID "Watcom"

/\* \_\_WATCOMC\_\_ = VVRR \*/

# define COMPILER\_VERSION\_MAJOR DEC(\_\_WATCOMC\_\_ / 100)

# define COMPILER\_VERSION\_MINOR DEC((\_\_WATCOMC\_\_ / 10) % 10)

# if (\_\_WATCOMC\_\_ % 10) > 0

# define COMPILER\_VERSION\_PATCH DEC(\_\_WATCOMC\_\_ % 10)

# endif

#elif defined(\_\_WATCOMC\_\_)

# define COMPILER\_ID "OpenWatcom"

/\* \_\_WATCOMC\_\_ = VVRP + 1100 \*/

# define COMPILER\_VERSION\_MAJOR DEC((\_\_WATCOMC\_\_ - 1100) / 100)

# define COMPILER\_VERSION\_MINOR DEC((\_\_WATCOMC\_\_ / 10) % 10)

# if (\_\_WATCOMC\_\_ % 10) > 0

# define COMPILER\_VERSION\_PATCH DEC(\_\_WATCOMC\_\_ % 10)

# endif

#elif defined(\_\_SUNPRO\_C)

# define COMPILER\_ID "SunPro"

# if \_\_SUNPRO\_C >= 0x5100

/\* \_\_SUNPRO\_C = 0xVRRP \*/

# define COMPILER\_VERSION\_MAJOR HEX(\_\_SUNPRO\_C>>12)

# define COMPILER\_VERSION\_MINOR HEX(\_\_SUNPRO\_C>>4 & 0xFF)

# define COMPILER\_VERSION\_PATCH HEX(\_\_SUNPRO\_C & 0xF)

# else

/\* \_\_SUNPRO\_CC = 0xVRP \*/

# define COMPILER\_VERSION\_MAJOR HEX(\_\_SUNPRO\_C>>8)

# define COMPILER\_VERSION\_MINOR HEX(\_\_SUNPRO\_C>>4 & 0xF)

# define COMPILER\_VERSION\_PATCH HEX(\_\_SUNPRO\_C & 0xF)

# endif

#elif defined(\_\_HP\_cc)

# define COMPILER\_ID "HP"

/\* \_\_HP\_cc = VVRRPP \*/

# define COMPILER\_VERSION\_MAJOR DEC(\_\_HP\_cc/10000)

# define COMPILER\_VERSION\_MINOR DEC(\_\_HP\_cc/100 % 100)

# define COMPILER\_VERSION\_PATCH DEC(\_\_HP\_cc % 100)

#elif defined(\_\_DECC)

# define COMPILER\_ID "Compaq"

/\* \_\_DECC\_VER = VVRRTPPPP \*/

# define COMPILER\_VERSION\_MAJOR DEC(\_\_DECC\_VER/10000000)

# define COMPILER\_VERSION\_MINOR DEC(\_\_DECC\_VER/100000 % 100)

# define COMPILER\_VERSION\_PATCH DEC(\_\_DECC\_VER % 10000)

#elif defined(\_\_IBMC\_\_) && defined(\_\_COMPILER\_VER\_\_)

# define COMPILER\_ID "zOS"

/\* \_\_IBMC\_\_ = VRP \*/

# define COMPILER\_VERSION\_MAJOR DEC(\_\_IBMC\_\_/100)

# define COMPILER\_VERSION\_MINOR DEC(\_\_IBMC\_\_/10 % 10)

# define COMPILER\_VERSION\_PATCH DEC(\_\_IBMC\_\_ % 10)

#elif defined(\_\_open\_xl\_\_) && defined(\_\_clang\_\_)

# define COMPILER\_ID "IBMClang"

# define COMPILER\_VERSION\_MAJOR DEC(\_\_open\_xl\_version\_\_)

# define COMPILER\_VERSION\_MINOR DEC(\_\_open\_xl\_release\_\_)

# define COMPILER\_VERSION\_PATCH DEC(\_\_open\_xl\_modification\_\_)

# define COMPILER\_VERSION\_TWEAK DEC(\_\_open\_xl\_ptf\_fix\_level\_\_)

#elif defined(\_\_ibmxl\_\_) && defined(\_\_clang\_\_)

# define COMPILER\_ID "XLClang"

# define COMPILER\_VERSION\_MAJOR DEC(\_\_ibmxl\_version\_\_)

# define COMPILER\_VERSION\_MINOR DEC(\_\_ibmxl\_release\_\_)

# define COMPILER\_VERSION\_PATCH DEC(\_\_ibmxl\_modification\_\_)

# define COMPILER\_VERSION\_TWEAK DEC(\_\_ibmxl\_ptf\_fix\_level\_\_)

#elif defined(\_\_IBMC\_\_) && !defined(\_\_COMPILER\_VER\_\_) && \_\_IBMC\_\_ >= 800

# define COMPILER\_ID "XL"

/\* \_\_IBMC\_\_ = VRP \*/

# define COMPILER\_VERSION\_MAJOR DEC(\_\_IBMC\_\_/100)

# define COMPILER\_VERSION\_MINOR DEC(\_\_IBMC\_\_/10 % 10)

# define COMPILER\_VERSION\_PATCH DEC(\_\_IBMC\_\_ % 10)

#elif defined(\_\_IBMC\_\_) && !defined(\_\_COMPILER\_VER\_\_) && \_\_IBMC\_\_ < 800

# define COMPILER\_ID "VisualAge"

/\* \_\_IBMC\_\_ = VRP \*/

# define COMPILER\_VERSION\_MAJOR DEC(\_\_IBMC\_\_/100)

# define COMPILER\_VERSION\_MINOR DEC(\_\_IBMC\_\_/10 % 10)

# define COMPILER\_VERSION\_PATCH DEC(\_\_IBMC\_\_ % 10)

#elif defined(\_\_NVCOMPILER)

# define COMPILER\_ID "NVHPC"

# define COMPILER\_VERSION\_MAJOR DEC(\_\_NVCOMPILER\_MAJOR\_\_)

# define COMPILER\_VERSION\_MINOR DEC(\_\_NVCOMPILER\_MINOR\_\_)

# if defined(\_\_NVCOMPILER\_PATCHLEVEL\_\_)

# define COMPILER\_VERSION\_PATCH DEC(\_\_NVCOMPILER\_PATCHLEVEL\_\_)

# endif

#elif defined(\_\_PGI)

# define COMPILER\_ID "PGI"

# define COMPILER\_VERSION\_MAJOR DEC(\_\_PGIC\_\_)

# define COMPILER\_VERSION\_MINOR DEC(\_\_PGIC\_MINOR\_\_)

# if defined(\_\_PGIC\_PATCHLEVEL\_\_)

# define COMPILER\_VERSION\_PATCH DEC(\_\_PGIC\_PATCHLEVEL\_\_)

# endif

#elif defined(\_\_clang\_\_) && defined(\_\_cray\_\_)

# define COMPILER\_ID "CrayClang"

# define COMPILER\_VERSION\_MAJOR DEC(\_\_cray\_major\_\_)

# define COMPILER\_VERSION\_MINOR DEC(\_\_cray\_minor\_\_)

# define COMPILER\_VERSION\_PATCH DEC(\_\_cray\_patchlevel\_\_)

# define COMPILER\_VERSION\_INTERNAL\_STR \_\_clang\_version\_\_

#elif defined(\_CRAYC)

# define COMPILER\_ID "Cray"

# define COMPILER\_VERSION\_MAJOR DEC(\_RELEASE\_MAJOR)

# define COMPILER\_VERSION\_MINOR DEC(\_RELEASE\_MINOR)

#elif defined(\_\_TI\_COMPILER\_VERSION\_\_)

# define COMPILER\_ID "TI"

/\* \_\_TI\_COMPILER\_VERSION\_\_ = VVVRRRPPP \*/

# define COMPILER\_VERSION\_MAJOR DEC(\_\_TI\_COMPILER\_VERSION\_\_/1000000)

# define COMPILER\_VERSION\_MINOR DEC(\_\_TI\_COMPILER\_VERSION\_\_/1000 % 1000)

# define COMPILER\_VERSION\_PATCH DEC(\_\_TI\_COMPILER\_VERSION\_\_ % 1000)

#elif defined(\_\_CLANG\_FUJITSU)

# define COMPILER\_ID "FujitsuClang"

# define COMPILER\_VERSION\_MAJOR DEC(\_\_FCC\_major\_\_)

# define COMPILER\_VERSION\_MINOR DEC(\_\_FCC\_minor\_\_)

# define COMPILER\_VERSION\_PATCH DEC(\_\_FCC\_patchlevel\_\_)

# define COMPILER\_VERSION\_INTERNAL\_STR \_\_clang\_version\_\_

#elif defined(\_\_FUJITSU)

# define COMPILER\_ID "Fujitsu"

# if defined(\_\_FCC\_version\_\_)

# define COMPILER\_VERSION \_\_FCC\_version\_\_

# elif defined(\_\_FCC\_major\_\_)

# define COMPILER\_VERSION\_MAJOR DEC(\_\_FCC\_major\_\_)

# define COMPILER\_VERSION\_MINOR DEC(\_\_FCC\_minor\_\_)

# define COMPILER\_VERSION\_PATCH DEC(\_\_FCC\_patchlevel\_\_)

# endif

# if defined(\_\_fcc\_version)

# define COMPILER\_VERSION\_INTERNAL DEC(\_\_fcc\_version)

# elif defined(\_\_FCC\_VERSION)

# define COMPILER\_VERSION\_INTERNAL DEC(\_\_FCC\_VERSION)

# endif

#elif defined(\_\_ghs\_\_)

# define COMPILER\_ID "GHS"

/\* \_\_GHS\_VERSION\_NUMBER = VVVVRP \*/

# ifdef \_\_GHS\_VERSION\_NUMBER

# define COMPILER\_VERSION\_MAJOR DEC(\_\_GHS\_VERSION\_NUMBER / 100)

# define COMPILER\_VERSION\_MINOR DEC(\_\_GHS\_VERSION\_NUMBER / 10 % 10)

# define COMPILER\_VERSION\_PATCH DEC(\_\_GHS\_VERSION\_NUMBER % 10)

# endif

#elif defined(\_\_TASKING\_\_)

# define COMPILER\_ID "Tasking"

# define COMPILER\_VERSION\_MAJOR DEC(\_\_VERSION\_\_/1000)

# define COMPILER\_VERSION\_MINOR DEC(\_\_VERSION\_\_ % 100)

# define COMPILER\_VERSION\_INTERNAL DEC(\_\_VERSION\_\_)

#elif defined(\_\_ORANGEC\_\_)

# define COMPILER\_ID "OrangeC"

# define COMPILER\_VERSION\_MAJOR DEC(\_\_ORANGEC\_MAJOR\_\_)

# define COMPILER\_VERSION\_MINOR DEC(\_\_ORANGEC\_MINOR\_\_)

# define COMPILER\_VERSION\_PATCH DEC(\_\_ORANGEC\_PATCHLEVEL\_\_)

#elif defined(\_\_TINYC\_\_)

# define COMPILER\_ID "TinyCC"

#elif defined(\_\_BCC\_\_)

# define COMPILER\_ID "Bruce"

#elif defined(\_\_SCO\_VERSION\_\_)

# define COMPILER\_ID "SCO"

#elif defined(\_\_ARMCC\_VERSION) && !defined(\_\_clang\_\_)

# define COMPILER\_ID "ARMCC"

#if \_\_ARMCC\_VERSION >= 1000000

/\* \_\_ARMCC\_VERSION = VRRPPPP \*/

# define COMPILER\_VERSION\_MAJOR DEC(\_\_ARMCC\_VERSION/1000000)

# define COMPILER\_VERSION\_MINOR DEC(\_\_ARMCC\_VERSION/10000 % 100)

# define COMPILER\_VERSION\_PATCH DEC(\_\_ARMCC\_VERSION % 10000)

#else

/\* \_\_ARMCC\_VERSION = VRPPPP \*/

# define COMPILER\_VERSION\_MAJOR DEC(\_\_ARMCC\_VERSION/100000)

# define COMPILER\_VERSION\_MINOR DEC(\_\_ARMCC\_VERSION/10000 % 10)

# define COMPILER\_VERSION\_PATCH DEC(\_\_ARMCC\_VERSION % 10000)

#endif

#elif defined(\_\_clang\_\_) && defined(\_\_apple\_build\_version\_\_)

# define COMPILER\_ID "AppleClang"

# if defined(\_MSC\_VER)

# define SIMULATE\_ID "MSVC"

# endif

# define COMPILER\_VERSION\_MAJOR DEC(\_\_clang\_major\_\_)

# define COMPILER\_VERSION\_MINOR DEC(\_\_clang\_minor\_\_)

# define COMPILER\_VERSION\_PATCH DEC(\_\_clang\_patchlevel\_\_)

# if defined(\_MSC\_VER)

/\* \_MSC\_VER = VVRR \*/

# define SIMULATE\_VERSION\_MAJOR DEC(\_MSC\_VER / 100)

# define SIMULATE\_VERSION\_MINOR DEC(\_MSC\_VER % 100)

# endif

# define COMPILER\_VERSION\_TWEAK DEC(\_\_apple\_build\_version\_\_)

#elif defined(\_\_clang\_\_) && defined(\_\_ARMCOMPILER\_VERSION)

# define COMPILER\_ID "ARMClang"

# define COMPILER\_VERSION\_MAJOR DEC(\_\_ARMCOMPILER\_VERSION/1000000)

# define COMPILER\_VERSION\_MINOR DEC(\_\_ARMCOMPILER\_VERSION/10000 % 100)

# define COMPILER\_VERSION\_PATCH DEC(\_\_ARMCOMPILER\_VERSION/100 % 100)

# define COMPILER\_VERSION\_INTERNAL DEC(\_\_ARMCOMPILER\_VERSION)

#elif defined(\_\_clang\_\_) && defined(\_\_ti\_\_)

# define COMPILER\_ID "TIClang"

# define COMPILER\_VERSION\_MAJOR DEC(\_\_ti\_major\_\_)

# define COMPILER\_VERSION\_MINOR DEC(\_\_ti\_minor\_\_)

# define COMPILER\_VERSION\_PATCH DEC(\_\_ti\_patchlevel\_\_)

# define COMPILER\_VERSION\_INTERNAL DEC(\_\_ti\_version\_\_)

#elif defined(\_\_clang\_\_)

# define COMPILER\_ID "Clang"

# if defined(\_MSC\_VER)

# define SIMULATE\_ID "MSVC"

# endif

# define COMPILER\_VERSION\_MAJOR DEC(\_\_clang\_major\_\_)

# define COMPILER\_VERSION\_MINOR DEC(\_\_clang\_minor\_\_)

# define COMPILER\_VERSION\_PATCH DEC(\_\_clang\_patchlevel\_\_)

# if defined(\_MSC\_VER)

/\* \_MSC\_VER = VVRR \*/

# define SIMULATE\_VERSION\_MAJOR DEC(\_MSC\_VER / 100)

# define SIMULATE\_VERSION\_MINOR DEC(\_MSC\_VER % 100)

# endif

#elif defined(\_\_LCC\_\_) && (defined(\_\_GNUC\_\_) || defined(\_\_GNUG\_\_) || defined(\_\_MCST\_\_))

# define COMPILER\_ID "LCC"

# define COMPILER\_VERSION\_MAJOR DEC(\_\_LCC\_\_ / 100)

# define COMPILER\_VERSION\_MINOR DEC(\_\_LCC\_\_ % 100)

# if defined(\_\_LCC\_MINOR\_\_)

# define COMPILER\_VERSION\_PATCH DEC(\_\_LCC\_MINOR\_\_)

# endif

# if defined(\_\_GNUC\_\_) && defined(\_\_GNUC\_MINOR\_\_)

# define SIMULATE\_ID "GNU"

# define SIMULATE\_VERSION\_MAJOR DEC(\_\_GNUC\_\_)

# define SIMULATE\_VERSION\_MINOR DEC(\_\_GNUC\_MINOR\_\_)

# if defined(\_\_GNUC\_PATCHLEVEL\_\_)

# define SIMULATE\_VERSION\_PATCH DEC(\_\_GNUC\_PATCHLEVEL\_\_)

# endif

# endif

#elif defined(\_\_GNUC\_\_)

# define COMPILER\_ID "GNU"

# define COMPILER\_VERSION\_MAJOR DEC(\_\_GNUC\_\_)

# if defined(\_\_GNUC\_MINOR\_\_)

# define COMPILER\_VERSION\_MINOR DEC(\_\_GNUC\_MINOR\_\_)

# endif

# if defined(\_\_GNUC\_PATCHLEVEL\_\_)

# define COMPILER\_VERSION\_PATCH DEC(\_\_GNUC\_PATCHLEVEL\_\_)

# endif

#elif defined(\_MSC\_VER)

# define COMPILER\_ID "MSVC"

/\* \_MSC\_VER = VVRR \*/

# define COMPILER\_VERSION\_MAJOR DEC(\_MSC\_VER / 100)

# define COMPILER\_VERSION\_MINOR DEC(\_MSC\_VER % 100)

# if defined(\_MSC\_FULL\_VER)

# if \_MSC\_VER >= 1400

/\* \_MSC\_FULL\_VER = VVRRPPPPP \*/

# define COMPILER\_VERSION\_PATCH DEC(\_MSC\_FULL\_VER % 100000)

# else

/\* \_MSC\_FULL\_VER = VVRRPPPP \*/

# define COMPILER\_VERSION\_PATCH DEC(\_MSC\_FULL\_VER % 10000)

# endif

# endif

# if defined(\_MSC\_BUILD)

# define COMPILER\_VERSION\_TWEAK DEC(\_MSC\_BUILD)

# endif

#elif defined(\_ADI\_COMPILER)

# define COMPILER\_ID "ADSP"

#if defined(\_\_VERSIONNUM\_\_)

/\* \_\_VERSIONNUM\_\_ = 0xVVRRPPTT \*/

# define COMPILER\_VERSION\_MAJOR DEC(\_\_VERSIONNUM\_\_ >> 24 & 0xFF)

# define COMPILER\_VERSION\_MINOR DEC(\_\_VERSIONNUM\_\_ >> 16 & 0xFF)

# define COMPILER\_VERSION\_PATCH DEC(\_\_VERSIONNUM\_\_ >> 8 & 0xFF)

# define COMPILER\_VERSION\_TWEAK DEC(\_\_VERSIONNUM\_\_ & 0xFF)

#endif

#elif defined(\_\_IAR\_SYSTEMS\_ICC\_\_) || defined(\_\_IAR\_SYSTEMS\_ICC)

# define COMPILER\_ID "IAR"

# if defined(\_\_VER\_\_) && defined(\_\_ICCARM\_\_)

# define COMPILER\_VERSION\_MAJOR DEC((\_\_VER\_\_) / 1000000)

# define COMPILER\_VERSION\_MINOR DEC(((\_\_VER\_\_) / 1000) % 1000)

# define COMPILER\_VERSION\_PATCH DEC((\_\_VER\_\_) % 1000)

# define COMPILER\_VERSION\_INTERNAL DEC(\_\_IAR\_SYSTEMS\_ICC\_\_)

# elif defined(\_\_VER\_\_) && (defined(\_\_ICCAVR\_\_) || defined(\_\_ICCRX\_\_) || defined(\_\_ICCRH850\_\_) || defined(\_\_ICCRL78\_\_) || defined(\_\_ICC430\_\_) || defined(\_\_ICCRISCV\_\_) || defined(\_\_ICCV850\_\_) || defined(\_\_ICC8051\_\_) || defined(\_\_ICCSTM8\_\_))

# define COMPILER\_VERSION\_MAJOR DEC((\_\_VER\_\_) / 100)

# define COMPILER\_VERSION\_MINOR DEC((\_\_VER\_\_) - (((\_\_VER\_\_) / 100)\*100))

# define COMPILER\_VERSION\_PATCH DEC(\_\_SUBVERSION\_\_)

# define COMPILER\_VERSION\_INTERNAL DEC(\_\_IAR\_SYSTEMS\_ICC\_\_)

# endif

#elif defined(\_\_SDCC\_VERSION\_MAJOR) || defined(SDCC)

# define COMPILER\_ID "SDCC"

# if defined(\_\_SDCC\_VERSION\_MAJOR)

# define COMPILER\_VERSION\_MAJOR DEC(\_\_SDCC\_VERSION\_MAJOR)

# define COMPILER\_VERSION\_MINOR DEC(\_\_SDCC\_VERSION\_MINOR)

# define COMPILER\_VERSION\_PATCH DEC(\_\_SDCC\_VERSION\_PATCH)

# else

/\* SDCC = VRP \*/

# define COMPILER\_VERSION\_MAJOR DEC(SDCC/100)

# define COMPILER\_VERSION\_MINOR DEC(SDCC/10 % 10)

# define COMPILER\_VERSION\_PATCH DEC(SDCC % 10)

# endif

/\* These compilers are either not known or too old to define an

identification macro. Try to identify the platform and guess that

it is the native compiler. \*/

#elif defined(\_\_hpux) || defined(\_\_hpua)

# define COMPILER\_ID "HP"

#else /\* unknown compiler \*/

# define COMPILER\_ID ""

#endif

/\* Construct the string literal in pieces to prevent the source from

getting matched. Store it in a pointer rather than an array

because some compilers will just produce instructions to fill the

array rather than assigning a pointer to a static array. \*/

char const\* info\_compiler = "INFO" ":" "compiler[" COMPILER\_ID "]";

#ifdef SIMULATE\_ID

char const\* info\_simulate = "INFO" ":" "simulate[" SIMULATE\_ID "]";

#endif

#ifdef \_\_QNXNTO\_\_

char const\* qnxnto = "INFO" ":" "qnxnto[]";

#endif

#if defined(\_\_CRAYXT\_COMPUTE\_LINUX\_TARGET)

char const \*info\_cray = "INFO" ":" "compiler\_wrapper[CrayPrgEnv]";

#endif

#define STRINGIFY\_HELPER(X) #X

#define STRINGIFY(X) STRINGIFY\_HELPER(X)

/\* Identify known platforms by name. \*/

#if defined(\_\_linux) || defined(\_\_linux\_\_) || defined(linux)

# define PLATFORM\_ID "Linux"

#elif defined(\_\_MSYS\_\_)

# define PLATFORM\_ID "MSYS"

#elif defined(\_\_CYGWIN\_\_)

# define PLATFORM\_ID "Cygwin"

#elif defined(\_\_MINGW32\_\_)

# define PLATFORM\_ID "MinGW"

#elif defined(\_\_APPLE\_\_)

# define PLATFORM\_ID "Darwin"

#elif defined(\_WIN32) || defined(\_\_WIN32\_\_) || defined(WIN32)

# define PLATFORM\_ID "Windows"

#elif defined(\_\_FreeBSD\_\_) || defined(\_\_FreeBSD)

# define PLATFORM\_ID "FreeBSD"

#elif defined(\_\_NetBSD\_\_) || defined(\_\_NetBSD)

# define PLATFORM\_ID "NetBSD"

#elif defined(\_\_OpenBSD\_\_) || defined(\_\_OPENBSD)

# define PLATFORM\_ID "OpenBSD"

#elif defined(\_\_sun) || defined(sun)

# define PLATFORM\_ID "SunOS"

#elif defined(\_AIX) || defined(\_\_AIX) || defined(\_\_AIX\_\_) || defined(\_\_aix) || defined(\_\_aix\_\_)

# define PLATFORM\_ID "AIX"

#elif defined(\_\_hpux) || defined(\_\_hpux\_\_)

# define PLATFORM\_ID "HP-UX"

#elif defined(\_\_HAIKU\_\_)

# define PLATFORM\_ID "Haiku"

#elif defined(\_\_BeOS) || defined(\_\_BEOS\_\_) || defined(\_BEOS)

# define PLATFORM\_ID "BeOS"

#elif defined(\_\_QNX\_\_) || defined(\_\_QNXNTO\_\_)

# define PLATFORM\_ID "QNX"

#elif defined(\_\_tru64) || defined(\_tru64) || defined(\_\_TRU64\_\_)

# define PLATFORM\_ID "Tru64"

#elif defined(\_\_riscos) || defined(\_\_riscos\_\_)

# define PLATFORM\_ID "RISCos"

#elif defined(\_\_sinix) || defined(\_\_sinix\_\_) || defined(\_\_SINIX\_\_)

# define PLATFORM\_ID "SINIX"

#elif defined(\_\_UNIX\_SV\_\_)

# define PLATFORM\_ID "UNIX\_SV"

#elif defined(\_\_bsdos\_\_)

# define PLATFORM\_ID "BSDOS"

#elif defined(\_MPRAS) || defined(MPRAS)

# define PLATFORM\_ID "MP-RAS"

#elif defined(\_\_osf) || defined(\_\_osf\_\_)

# define PLATFORM\_ID "OSF1"

#elif defined(\_SCO\_SV) || defined(SCO\_SV) || defined(sco\_sv)

# define PLATFORM\_ID "SCO\_SV"

#elif defined(\_\_ultrix) || defined(\_\_ultrix\_\_) || defined(\_ULTRIX)

# define PLATFORM\_ID "ULTRIX"

#elif defined(\_\_XENIX\_\_) || defined(\_XENIX) || defined(XENIX)

# define PLATFORM\_ID "Xenix"

#elif defined(\_\_WATCOMC\_\_)

# if defined(\_\_LINUX\_\_)

# define PLATFORM\_ID "Linux"

# elif defined(\_\_DOS\_\_)

# define PLATFORM\_ID "DOS"

# elif defined(\_\_OS2\_\_)

# define PLATFORM\_ID "OS2"

# elif defined(\_\_WINDOWS\_\_)

# define PLATFORM\_ID "Windows3x"

# elif defined(\_\_VXWORKS\_\_)

# define PLATFORM\_ID "VxWorks"

# else /\* unknown platform \*/

# define PLATFORM\_ID

# endif

#elif defined(\_\_INTEGRITY)

# if defined(INT\_178B)

# define PLATFORM\_ID "Integrity178"

# else /\* regular Integrity \*/

# define PLATFORM\_ID "Integrity"

# endif

# elif defined(\_ADI\_COMPILER)

# define PLATFORM\_ID "ADSP"

#else /\* unknown platform \*/

# define PLATFORM\_ID

#endif

/\* For windows compilers MSVC and Intel we can determine

the architecture of the compiler being used. This is because

the compilers do not have flags that can change the architecture,

but rather depend on which compiler is being used

\*/

#if defined(\_WIN32) && defined(\_MSC\_VER)

# if defined(\_M\_IA64)

# define ARCHITECTURE\_ID "IA64"

# elif defined(\_M\_ARM64EC)

# define ARCHITECTURE\_ID "ARM64EC"

# elif defined(\_M\_X64) || defined(\_M\_AMD64)

# define ARCHITECTURE\_ID "x64"

# elif defined(\_M\_IX86)

# define ARCHITECTURE\_ID "X86"

# elif defined(\_M\_ARM64)

# define ARCHITECTURE\_ID "ARM64"

# elif defined(\_M\_ARM)

# if \_M\_ARM == 4

# define ARCHITECTURE\_ID "ARMV4I"

# elif \_M\_ARM == 5

# define ARCHITECTURE\_ID "ARMV5I"

# else

# define ARCHITECTURE\_ID "ARMV" STRINGIFY(\_M\_ARM)

# endif

# elif defined(\_M\_MIPS)

# define ARCHITECTURE\_ID "MIPS"

# elif defined(\_M\_SH)

# define ARCHITECTURE\_ID "SHx"

# else /\* unknown architecture \*/

# define ARCHITECTURE\_ID ""

# endif

#elif defined(\_\_WATCOMC\_\_)

# if defined(\_M\_I86)

# define ARCHITECTURE\_ID "I86"

# elif defined(\_M\_IX86)

# define ARCHITECTURE\_ID "X86"

# else /\* unknown architecture \*/

# define ARCHITECTURE\_ID ""

# endif

#elif defined(\_\_IAR\_SYSTEMS\_ICC\_\_) || defined(\_\_IAR\_SYSTEMS\_ICC)

# if defined(\_\_ICCARM\_\_)

# define ARCHITECTURE\_ID "ARM"

# elif defined(\_\_ICCRX\_\_)

# define ARCHITECTURE\_ID "RX"

# elif defined(\_\_ICCRH850\_\_)

# define ARCHITECTURE\_ID "RH850"

# elif defined(\_\_ICCRL78\_\_)

# define ARCHITECTURE\_ID "RL78"

# elif defined(\_\_ICCRISCV\_\_)

# define ARCHITECTURE\_ID "RISCV"

# elif defined(\_\_ICCAVR\_\_)

# define ARCHITECTURE\_ID "AVR"

# elif defined(\_\_ICC430\_\_)

# define ARCHITECTURE\_ID "MSP430"

# elif defined(\_\_ICCV850\_\_)

# define ARCHITECTURE\_ID "V850"

# elif defined(\_\_ICC8051\_\_)

# define ARCHITECTURE\_ID "8051"

# elif defined(\_\_ICCSTM8\_\_)

# define ARCHITECTURE\_ID "STM8"

# else /\* unknown architecture \*/

# define ARCHITECTURE\_ID ""

# endif

#elif defined(\_\_ghs\_\_)

# if defined(\_\_PPC64\_\_)

# define ARCHITECTURE\_ID "PPC64"

# elif defined(\_\_ppc\_\_)

# define ARCHITECTURE\_ID "PPC"

# elif defined(\_\_ARM\_\_)

# define ARCHITECTURE\_ID "ARM"

# elif defined(\_\_x86\_64\_\_)

# define ARCHITECTURE\_ID "x64"

# elif defined(\_\_i386\_\_)

# define ARCHITECTURE\_ID "X86"

# else /\* unknown architecture \*/

# define ARCHITECTURE\_ID ""

# endif

#elif defined(\_\_clang\_\_) && defined(\_\_ti\_\_)

# if defined(\_\_ARM\_ARCH)

# define ARCHITECTURE\_ID "Arm"

# else /\* unknown architecture \*/

# define ARCHITECTURE\_ID ""

# endif

#elif defined(\_\_TI\_COMPILER\_VERSION\_\_)

# if defined(\_\_TI\_ARM\_\_)

# define ARCHITECTURE\_ID "ARM"

# elif defined(\_\_MSP430\_\_)

# define ARCHITECTURE\_ID "MSP430"

# elif defined(\_\_TMS320C28XX\_\_)

# define ARCHITECTURE\_ID "TMS320C28x"

# elif defined(\_\_TMS320C6X\_\_) || defined(\_TMS320C6X)

# define ARCHITECTURE\_ID "TMS320C6x"

# else /\* unknown architecture \*/

# define ARCHITECTURE\_ID ""

# endif

# elif defined(\_\_ADSPSHARC\_\_)

# define ARCHITECTURE\_ID "SHARC"

# elif defined(\_\_ADSPBLACKFIN\_\_)

# define ARCHITECTURE\_ID "Blackfin"

#elif defined(\_\_TASKING\_\_)

# if defined(\_\_CTC\_\_) || defined(\_\_CPTC\_\_)

# define ARCHITECTURE\_ID "TriCore"

# elif defined(\_\_CMCS\_\_)

# define ARCHITECTURE\_ID "MCS"

# elif defined(\_\_CARM\_\_)

# define ARCHITECTURE\_ID "ARM"

# elif defined(\_\_CARC\_\_)

# define ARCHITECTURE\_ID "ARC"

# elif defined(\_\_C51\_\_)

# define ARCHITECTURE\_ID "8051"

# elif defined(\_\_CPCP\_\_)

# define ARCHITECTURE\_ID "PCP"

# else

# define ARCHITECTURE\_ID ""

# endif

#else

# define ARCHITECTURE\_ID

#endif

/\* Convert integer to decimal digit literals. \*/

#define DEC(n) \

('0' + (((n) / 10000000)%10)), \

('0' + (((n) / 1000000)%10)), \

('0' + (((n) / 100000)%10)), \

('0' + (((n) / 10000)%10)), \

('0' + (((n) / 1000)%10)), \

('0' + (((n) / 100)%10)), \

('0' + (((n) / 10)%10)), \

('0' + ((n) % 10))

/\* Convert integer to hex digit literals. \*/

#define HEX(n) \

('0' + ((n)>>28 & 0xF)), \

('0' + ((n)>>24 & 0xF)), \

('0' + ((n)>>20 & 0xF)), \

('0' + ((n)>>16 & 0xF)), \

('0' + ((n)>>12 & 0xF)), \

('0' + ((n)>>8 & 0xF)), \

('0' + ((n)>>4 & 0xF)), \

('0' + ((n) & 0xF))

/\* Construct a string literal encoding the version number. \*/

#ifdef COMPILER\_VERSION

char const\* info\_version = "INFO" ":" "compiler\_version[" COMPILER\_VERSION "]";

/\* Construct a string literal encoding the version number components. \*/

#elif defined(COMPILER\_VERSION\_MAJOR)

char const info\_version[] = {

'I', 'N', 'F', 'O', ':',

'c','o','m','p','i','l','e','r','\_','v','e','r','s','i','o','n','[',

COMPILER\_VERSION\_MAJOR,

# ifdef COMPILER\_VERSION\_MINOR

'.', COMPILER\_VERSION\_MINOR,

# ifdef COMPILER\_VERSION\_PATCH

'.', COMPILER\_VERSION\_PATCH,

# ifdef COMPILER\_VERSION\_TWEAK

'.', COMPILER\_VERSION\_TWEAK,

# endif

# endif

# endif

']','\0'};

#endif

/\* Construct a string literal encoding the internal version number. \*/

#ifdef COMPILER\_VERSION\_INTERNAL

char const info\_version\_internal[] = {

'I', 'N', 'F', 'O', ':',

'c','o','m','p','i','l','e','r','\_','v','e','r','s','i','o','n','\_',

'i','n','t','e','r','n','a','l','[',

COMPILER\_VERSION\_INTERNAL,']','\0'};

#elif defined(COMPILER\_VERSION\_INTERNAL\_STR)

char const\* info\_version\_internal = "INFO" ":" "compiler\_version\_internal[" COMPILER\_VERSION\_INTERNAL\_STR "]";

#endif

/\* Construct a string literal encoding the version number components. \*/

#ifdef SIMULATE\_VERSION\_MAJOR

char const info\_simulate\_version[] = {

'I', 'N', 'F', 'O', ':',

's','i','m','u','l','a','t','e','\_','v','e','r','s','i','o','n','[',

SIMULATE\_VERSION\_MAJOR,

# ifdef SIMULATE\_VERSION\_MINOR

'.', SIMULATE\_VERSION\_MINOR,

# ifdef SIMULATE\_VERSION\_PATCH

'.', SIMULATE\_VERSION\_PATCH,

# ifdef SIMULATE\_VERSION\_TWEAK

'.', SIMULATE\_VERSION\_TWEAK,

# endif

# endif

# endif

']','\0'};

#endif

/\* Construct the string literal in pieces to prevent the source from

getting matched. Store it in a pointer rather than an array

because some compilers will just produce instructions to fill the

array rather than assigning a pointer to a static array. \*/

char const\* info\_platform = "INFO" ":" "platform[" PLATFORM\_ID "]";

char const\* info\_arch = "INFO" ":" "arch[" ARCHITECTURE\_ID "]";

#define C\_STD\_99 199901L

#define C\_STD\_11 201112L

#define C\_STD\_17 201710L

#define C\_STD\_23 202311L

#ifdef \_\_STDC\_VERSION\_\_

# define C\_STD \_\_STDC\_VERSION\_\_

#endif

#if !defined(\_\_STDC\_\_) && !defined(\_\_clang\_\_)

# if defined(\_MSC\_VER) || defined(\_\_ibmxl\_\_) || defined(\_\_IBMC\_\_)

# define C\_VERSION "90"

# else

# define C\_VERSION

# endif

#elif C\_STD > C\_STD\_17

# define C\_VERSION "23"

#elif C\_STD > C\_STD\_11

# define C\_VERSION "17"

#elif C\_STD > C\_STD\_99

# define C\_VERSION "11"

#elif C\_STD >= C\_STD\_99

# define C\_VERSION "99"

#else

# define C\_VERSION "90"

#endif

const char\* info\_language\_standard\_default =

"INFO" ":" "standard\_default[" C\_VERSION "]";

const char\* info\_language\_extensions\_default = "INFO" ":" "extensions\_default["

#if (defined(\_\_clang\_\_) || defined(\_\_GNUC\_\_) || defined(\_\_xlC\_\_) || \

defined(\_\_TI\_COMPILER\_VERSION\_\_)) && \

!defined(\_\_STRICT\_ANSI\_\_)

"ON"

#else

"OFF"

#endif

"]";

/\*--------------------------------------------------------------------------\*/

#ifdef ID\_VOID\_MAIN

void main() {}

#else

# if defined(\_\_CLASSIC\_C\_\_)

int main(argc, argv) int argc; char \*argv[];

# else

int main(int argc, char\* argv[])

# endif

{

int require = 0;

require += info\_compiler[argc];

require += info\_platform[argc];

require += info\_arch[argc];

#ifdef COMPILER\_VERSION\_MAJOR

require += info\_version[argc];

#endif

#ifdef COMPILER\_VERSION\_INTERNAL

require += info\_version\_internal[argc];

#endif

#ifdef SIMULATE\_ID

require += info\_simulate[argc];

#endif

#ifdef SIMULATE\_VERSION\_MAJOR

require += info\_simulate\_version[argc];

#endif

#if defined(\_\_CRAYXT\_COMPUTE\_LINUX\_TARGET)

require += info\_cray[argc];

#endif

require += info\_language\_standard\_default[argc];

require += info\_language\_extensions\_default[argc];

(void)argv;

return require;

}

#endif