I'm not robot
reCAPTCHA
Continue

C 11 makefile template

The original GNU C Compiler (GCC) was developed by Richard Stallman, founder of project GNU. Richard Stallman founded the GNU project in 1984 to create a unix-like operating system as free software to promote freedom and collaboration between computer users and programmers. Formerly the GNU C Compiler, GCC has grown over time to support many languages, including C (gcc), C++ (g++), Objective-C, Objective-C++, Java (gcj), Fortran (gfortran), Ada (gnat), Go (gccgo), OpenMP, Cilk Plus, and OpenAcc. It is now called the GNU Compiler Collection. The main site for GCC . GCC 7.3. GCC, released in the current version 2018-01-25, is an important component of the GNU Toolchain for developing applications and writing operating systems. GNU Toolchain includes: GNU Compiler Collection (GCC): A compiler package that supports many languages, such as C/C++ and Objective-C/C++. GNU Make: is an automation tool for compiling and building applications. GNU Binutils: a package of dual auxiliary tools, including connector and assembler. GNU Debugging (GDB). GNU Autotools: A building system including Autoconf, Autoheader, Automake and Libtool. GNU Bison: a parser generator (similar to lex and yacc). GCC is portable and run on many operating platforms. GCC (and GNU Toolchain) are currently available in all Unixes. They are also moved to Windows (by Cygwin, MinGW, and MinGW-W64). GCC is also a cross compiler for producing executables on a different platform. GCC Versions Various versions of GCC include: GCC version 1 (1987): The first version that supported C. GCC version 2 (1992): Supports C++. GCC version 3 (2001): Improve optimization, including ECGS (Experimental GNU Compiler System). GCC version 4 (2005): GCC version 5 (2015): GCC Version 6 (2016): GCC Version 7 (2017): C++ Standard Support Has various C++ standards: C++98 C++11 (aka C++0x) C++14 (aka C++1y) C++17 (aka C++1z) C++2a (next planned standard in 2020) Default mode 6.1 before GCC versions C++98 and GCC 6.1 and above c++14. You can use command-line flag -std to explicitly specify the C++ standard. For example, -std=c++98 or -std=gnu++98 (C++98 with GNU extension) -std=c++11 or -std=gnu++11 (G Default mode with C++11) -std=c++14 or -std=gnu++14 (C++14 with GNU extensions), GCC 6.1 and above. -std=c++17 or -std=gnu++17 (C++17 with GNU extension), experimental. -std=c++2a or -std=gnu++2a (C++2a with GNU extensions), experimental. Installing GCC on Unixes GNU Toolchain, including GCC, is included in all Unixes. It is the standard compiler for most Unix-like operating systems. Open a GCC installation Terminal in Mac OS X and enter a gcc --version. If GCC is not installed, the system will be asked to install gcc. $ gcc --version ..... Goal: x86_64-apple-darwin14.5.0 Thread model: You can install GCC installation posix, either Cygwin GCC, MinGW GCC, or MinGW-W64 GCC on Windows. Read Cygwin is installed and Cygwin GCC: Cygwin is a Unix-like environment and command-line interface for Microsoft Windows. Cygwin is large and includes most unix tools and utilitys. Also included in the widely used Bash shell. MinGW: MinGW (Minimalist GNU for Windows) is a port of GNU Compiler Collection (GCC) and GNU Binutils for use in Windows. Also basically included in a Bourne shell (bash) msys (Minimal System). MinGW-W64: A MinGW fork that supports both 32-bit and 64-bit windows. There are many GCCs under Cygwin under Various GCCs Cygain/MinGW. To distinguish these variations, you need to understand the following: Windows/Intel uses the following sets of instructions: x86 is a 32-bit set of instructions; i868 x86 is a 32-bit improved version; x86_64 (or amd64) is a 64-bit set of instructions. 32-bit compilers/programs can be run on 32-bit or 64-bit (backward-compatible) Windows, but the 64-bit compiler can only run on 64-bit Windows. 64-bit compilers can produce a 32-bit or 64-bit target. If you use Cygwin's GCC, the destination can be local Windows or Cygwin. If the destination is local Windows, the code can be deployed and run under Windows. However, if cygwin is the target, to deploy it, cygwin must deploy the runtime environment (cygwin1.dll) to deploy. Cygwin is a Unix e-producer under Windows. MinGW-W64 Target supports both 32-bit and 64-bit native Windows target of 32/64-bit Native Windows MinGW-W64 (MinGW is available in a fork, . By selecting these packages you can install MinGW-W64 under Cygwin (under the devel category): mingw64-x86_64-gcc-core: 64-bit C compiler for native 64-bit Windows target. Executable x86_64-w64-mingw32-gcc. mingw64-x86_64-gcc-g++: 64-bit C++ compiler for native 64-bit Windows target. Executable x86_64-w64-mingw32-g++. mingw64-i686-gcc-core: 64-bit C compiler for native 32-bit Windows target. Executable i686-w64-mingw32-gcc. mingw64-i686-gcc-g++: 64-bit C++ compiler for native 32-bit Windows target. Executable i686-w64-mingw32-g++. Notes: I recommend installing mingw64-x86_64-gcc-core and mingw64-x86_64-gcc-g++ to provide native 64-bit Windows codes, but skip mingw64-i686-gcc-core and mingw64-i686-gcc-g++, unless you need to produce 32-bit Windows applications. To generate 64-bit native Windows code for JNI (Java Native Interface) in 64-bit Java, you must use x86_64-w64-mingw32-gcc or x86_64-w64-mingw32-g++. Run executable and check versions: $x86_64-w64-mingw32-gcc --versionx86_64-w64-mingw32-gcc (GCC) $6,4.0 x86_64-w64-mingw32-gcc -vUsing internal features. COLLECT_GCC=x86_64-w64-mingw32-gccCOLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-w64-mingw32/6.4.0/lto-wrapper.exeTarget: x86_64-w64-mingw32So: ..... Subject model: posixgcc version 6.4.0 (GCC) $ x86_64-w64-mingw32-g++ --versionx86_64-w64-mingw32-g++ (GCC) $6.4.0 i686-w64-mingw32-gcc --versioni686-w64-mingw32-gcc (GCC) $6.4.0 (GCC) 6.4.0 Other GCCs in other GCC packages in Cygwin are: gcc-core, gcc-g++: Basic 64-bit C/C++ compiler target 64-bit Cygwin. You probably need to install both of these packages. However, to deploy generated code, you must deploy the Cygwin runtime environment (cygwin1.dll the generated code. Cygwin is a Unix e-producer under Windows. cygwin32-gcc-core, cygwin32-gcc-g++: Obsoleted by the old 32-bit C/C++ compiler (gcc-code and gcc-g++?) for target 32-bit Cygwin. mingw-gcc-core, mingw-gcc-g++: Old MinGW 32-bit C/C++ compiler for 32-bit Windows (obsoleted by MinGW-W64 packets?). Post Installation Versions GCC version -version option: $gcc (GCC) $6.4.0 g++ --version g++ (GCC) 6.4.0 More details can be obtained with the -v option, for example, using $gcc-v Internal features. COLLECT_GCC=gccCOLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-pc-cygwin/6.4.0/lto-wrapper.exeTarget: x86_64-pc-cygwinStructed: ..... Subject model: posixgcc version 6.4.0 (GCC) $ g++ -vUsing internal features. COLLECT_GCC=g++COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-pc-cygwin/6.4.0/lto-wrapper.exeTarget: x86_64-pc-cygwinStructed: ..... Thread model: You can get the posixgcc version 6.4.0 (GCC) Help Guide with the --help option. For example, you can read GCC manual pages (or man pages) through the $gcc --help Man Pages Human program: $man gcc$ man g++ CMD or Bash can be difficult to read man pages under the shell. You can create a text file: $ man gcc | Col -b > gcc.txt Col utility backspace ribbon is required. (For Cygwin, Utils is available in the util-linux package.) Alternatively, you can look at an online man pages, for example, to be published. GCC man pages are kept under usr/share/man/man1. $ whereis gcc gcc: /usr/bin/gcc.exe /usr/lib/gcc/usr/share/man/man1/gcc.1.gz Getting started GNU C and C++ compiler are called gcc and g++, respectively. Compilation / Link is a simple C Program - hello.c Hello-world C program hello below.c: 1 2 3 4 5 6 7 #include <stdio.h> int main() { printf(Hello, world!); return 0; } Hello.c: > gcc hello.c Default output is called executable a.exe (Windows) or a.out (Unixes and Mac OS X). To run the program: > $ chmod a+x a.out$ ./a.out Notes for Unixes and Bash Shell: Bash shell does not contain the default PATH current working directory. Therefore, you must add the current path (./) to the command. (Windows automatically contains the current directory in PATH; whereas it does not contain Unixes - you must explicitly add the current directory to PATH.) You must also add a file extension, that is, ./a.out, if any. In Unixes, the output file can be a.out or simply a. You must also assign chmod a+x filename command (add a+x executable file mode +x to all users) as executable file mode (x) a.out. Use the -o option to specify the output file name: > gcc -o </stdio.h>hello.c > hello $gcc -o hellomerhaba.c $chmod a+x hello $ ./hello NOTE for Unixes: In Unixes, we usually skip the .exe file extension (it's for Windows only) and just name the executable output with the command gcc -o hello.c. You must assign executable file mode with chmod a+x hello command. Compile/Link a Simple C++ Program - hello.cpp 1 2 3 4 5 6 7 8 #include <iostream>namespace std; int main() { cout < hello,= world!=> < endl;= return= 0;= }= you= need= to= use= g++= to= compile= c++= program,= as= follows.= we= use= the= -o= option= to= specify= the= output= file= name.=> g++ -o hello.exe hello.cpp > hello $ g++ -o hello.cpp $ chmod a+x hello $ ./hello More GCC Compiler Options Several frequently used GCC compiler options include: $g++ -Wall -g -o Hello.exe Hello.cpp -o: output executable file name. -Wall: prints all warning messages. -g: GDB creates additional symbolic debugging information for use with debugging. Separate Assembly and Binding Compile the above command source file into the object file and connect with other object files and system libraries as executable in one step. You can separate the compile and link in two steps, such as: > g++ -c -Wall -g Hello.cpp > g++ -g -o Hello.exe Hello.o Options are: -c: Hello.o assembly to object file. By default, the object file has the same name as the source file with the .o extension (no need to specify the -o option). Link to other object files or libraries. When the input file object files are .o (the source file .cpp is .c binding is performed instead. GCC uses a separate connector program (called ld.exe) to perform the connection. Compile and Link Multiple Source Files: File1.cpp, file2.cpp. You can compile them all with a single command: > g++ -o myprog.exe file1.cpp file2.cpp However, we can usually compile each of the source files into the individual object file and link them together in the next stage. In this case, changes to one file do not require the re-compile of other files. > g++ -c file1.cpp > g++ -c file2.cpp > g++ -o myprog.exe file1.o file2.o Compile and Link C/C++ to a shared library (.dll windows, .so unixes), usage -shared option. For example, read the Java Local Interface. GCC Build Process GCC compiles a C/C++ program as executable in 4 steps, as shown in the diagram above. For example, a gcc - o hello.exe hello.c is performed as follows: Pre-processing: GNU C Pre-processor (cpp.exe), containing headers (#include), and expanding macros (#define). > cpp hello.c > hello.i The resulting intermediate file contains hello.i extended source code. Assembly: The compiler compiles preprocessed source code into assembly code for a particular processor. > gcc -S hello.i -S option, instead of object code to generate code. The resulting build file is hello.s. Assembly:</iostream> Montage:</iostream> (as.exe) converts assembly code to hello.o machine code in the object file. > as -o hello.o hello.s Linker: Finally, the connector (ld.exe) connects object code with library code to generate an executable file hello.exe. > ld -o hello.exe hello.o ... Library... You can see the detailed compilation process by enabling verbose mode (-v) -v (detailed). For example, > gcc -v -o hello.exe hello.c Macro (-D) Identification Macro (-D) To define a macro , you can use the -Dname option or the -Dname=value to define a macro with a value. If it contains spaces, the value must be added in double quotes. Header (.h), Static Libraries (.lib, .a), and Shared Library (.dll, .so) Static Library vs Shared Library are a collection of precomperated object files that can be linked to your programs through the connector. Examples of system functions such as Printf() and sqrt(). There are two types of external libraries: static library and shared library. The static library has a .a (archive file) file extension in Unixes or a .lib file extension in Windows. When your program is connected to a static library, the machine code of the external functions used in your program is copied that can be executed. Ar.exe can be created through the archive program. The shared library has a .so file extension in Unixes or .dll (dynamic link library) in Windows. When your program is linked to a shared library, only a small table is created in the executable library. Before starting executable operation, the operating system installs the machine code required for external functions - a process known as dynamic binding. Because a copy of the library can be shared between multiple programs, dynamic binding shrinks executable files and saves disk space. In addition, most operating systems save memory by allowing a copy of a shared library in memory to be used by all running programs. Shared library codes can be upgraded without the need to re-compile your program. Because of dynamic binding, GCC by default, if any, shared library links. You can list the contents of the library through the nm file name. Searching Header Files and Libraries (-I, -L, and -l) When compiling the program, the compiler needs header files to compile source code; the connector needs libraries to resolve external references from other object files or libraries. The compiler and connector cannot find headers/libraries unless you set the appropriate options that are not open for the user for the first time. For each header used in your source (#include instructions), the compiler searches for the so-called include-paths for that header. Include-paths are specified by the -Idir option (or environment variable CPATH). Because the file name of the header is known (for example, iostream.h, stdio.h), the compiler can only be Has. The connector can The library path is specified by the -Ldir option (the uppercase 'L' index path) (or LIBRARY_PATH the environment variable). You must also specify the library name. Libxxx.a library -lxxx option in Unixes (lowercase letter 'l' without lib and .a extension prenel). Provide the full name in Windows, such as -lxxx.lib. The connector must know both directories and library names. Therefore, two options must be specified. Try the default Internal Paths, Library paths, and Libraries cpp -v through the default included paths used on your system by the GNU C Pre-processor: > cpp -v ..... #include ... the search starts here: #include <... >search starts here: /usr/lib/gcc/x86_64-pc-cygwin/6.4.0/include /usr/include /usr/usr/lib/gcc/x86_64-pc-cygwin/6.4.0/.. /.. /.. /.. /lib/... /include/w32api Try running the assembly in detailed mode (-v) to examine the library paths (-L) and libraries (-l) used on your system: > gcc -v -o hello.exe hello.c..... -L/usr/lib/gcc/x86_64-pc-cygwin/6.. 4.0 -L/usr/x86_64-pc-cygwin/lib -L/usr/lib -L/lib -lgcc_s -lgcc -lcygwin -ladvapi32 -lshell32 -luser32 -lkernel32 Eclipse CDT: Eclipse on CDT, library paths and libraries, Under the Include tabs⇒ C/C++ ⇒ Paths and Symbols ⇒ Properties ⇒ right-clicking the project, Library Paths and Libraries. The settings apply only to the selected project. GCC Environment Variables GCC uses the following environment variables: PATH: To search for executables and run-time shared libraries (.dll, .so). CPATH: For search for include-paths for header. <dir> is searched after the paths specified in -I options. C_INCLUDE_PATH and CPLUS_INCLUDE_PATH can be used to specify headers C and C++ if a specific language is specified in pre-processing. LIBRARY_PATH: To search for library paths for link libraries. <dir> is searched after the paths specified in -L options. Compiled Files Review Utility You can use command -help to list the help menu for all GNU utility; or man command man to view pages. File Utility - Determine File Type Helper program can be used to view file object files and the type of executable files. For example, $gcc -c hello.c $ gcc -o hello.exe $ gcc -o hello.exe hello.o $ file hello.chello.c: C source, ASCII text, CRLF line terminators with $ file hello.o hello.o: data > file hello.o hello.o: data > file hello.exe hello.exe: PE32 executable (console) x86-64, MS windows nm utility program nm lists object files symbol table. For example, $nm hello.o 0000000000