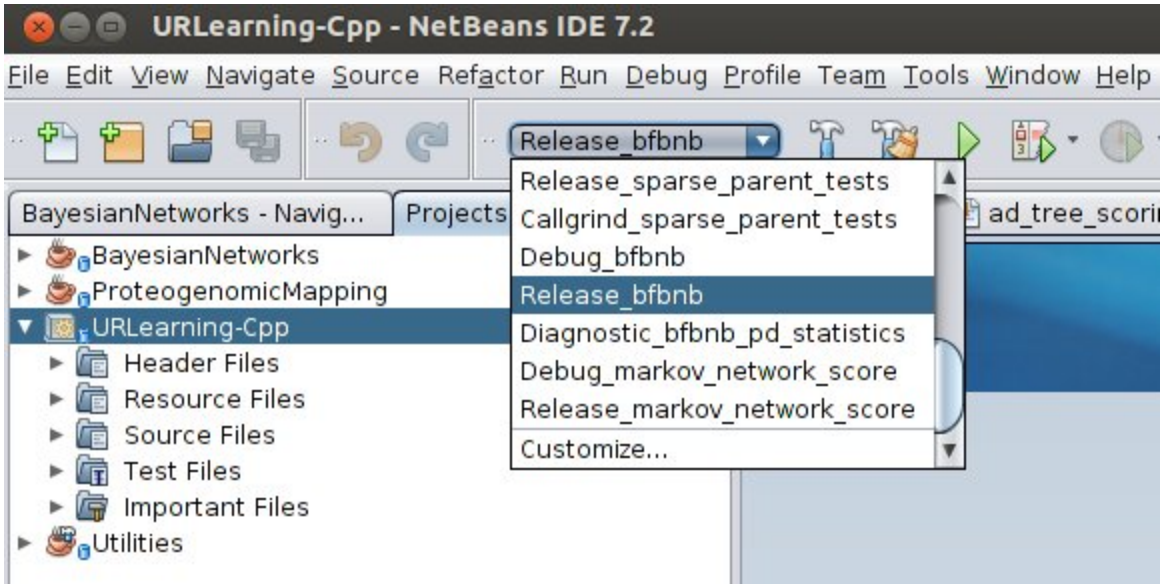
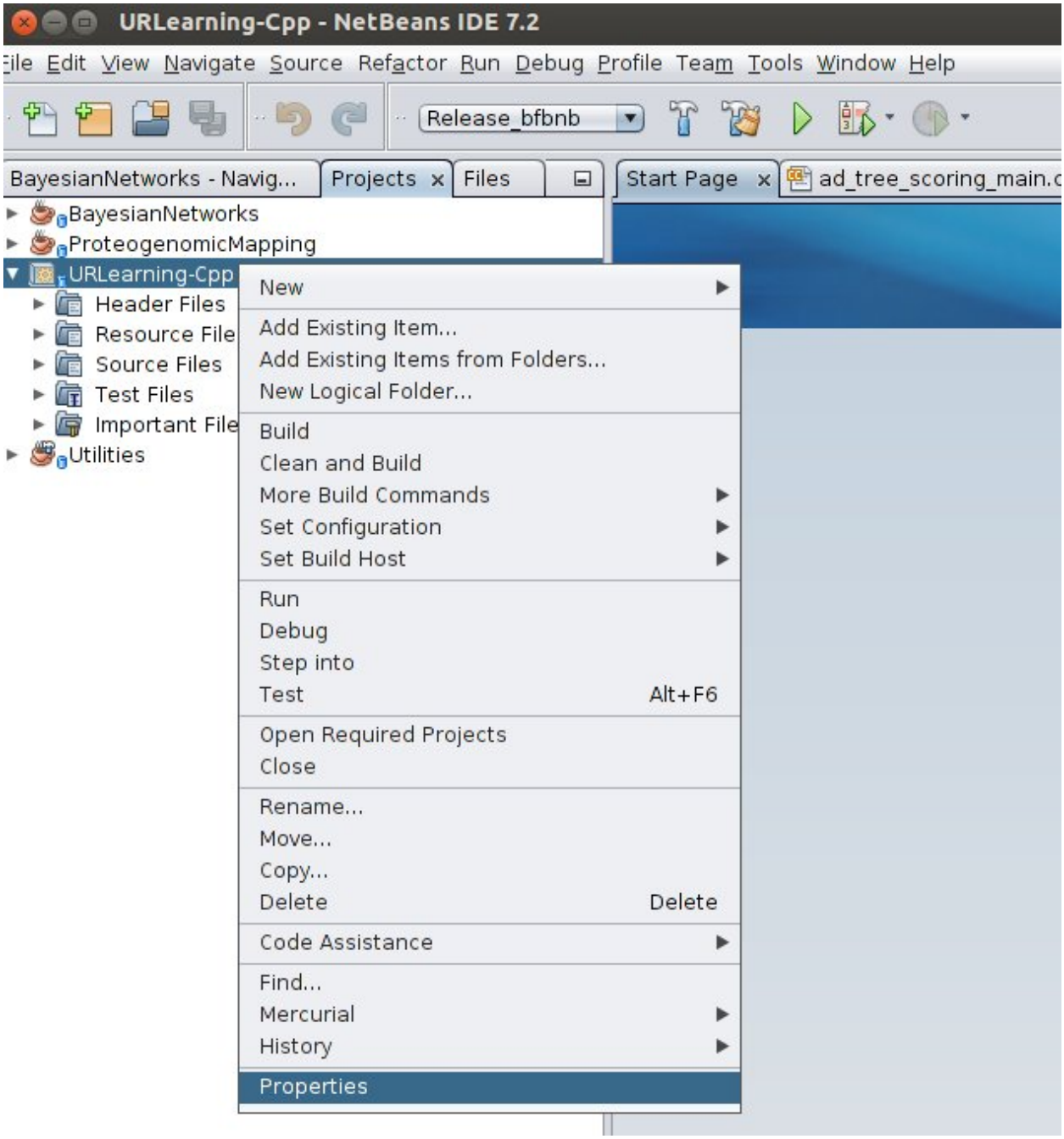
You will need to follow these steps for each configuration that needs to be compiled.

1. Select the configuration you wish to configure from the dropdown list.

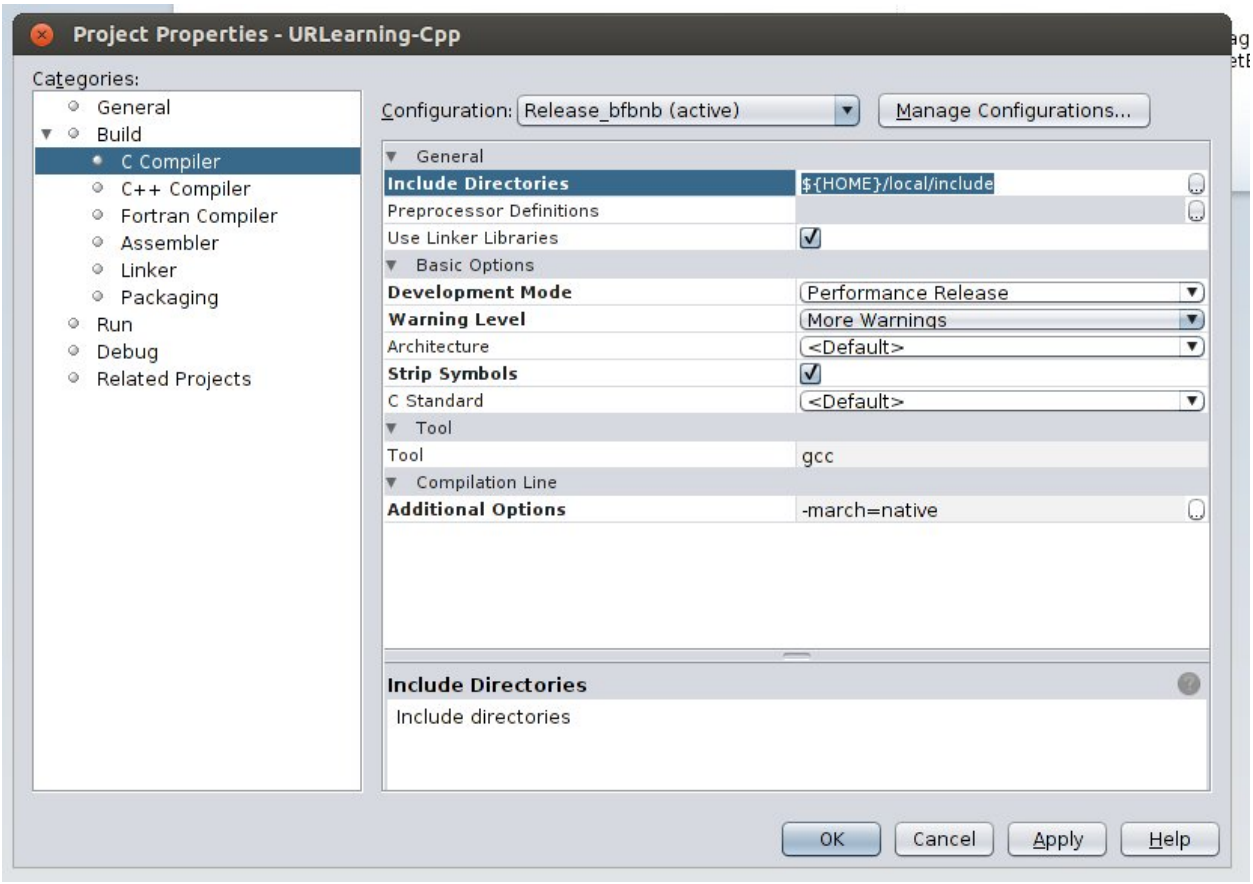


2. Right click on ‘URLearning-Cpp’ and select ‘Properties’.

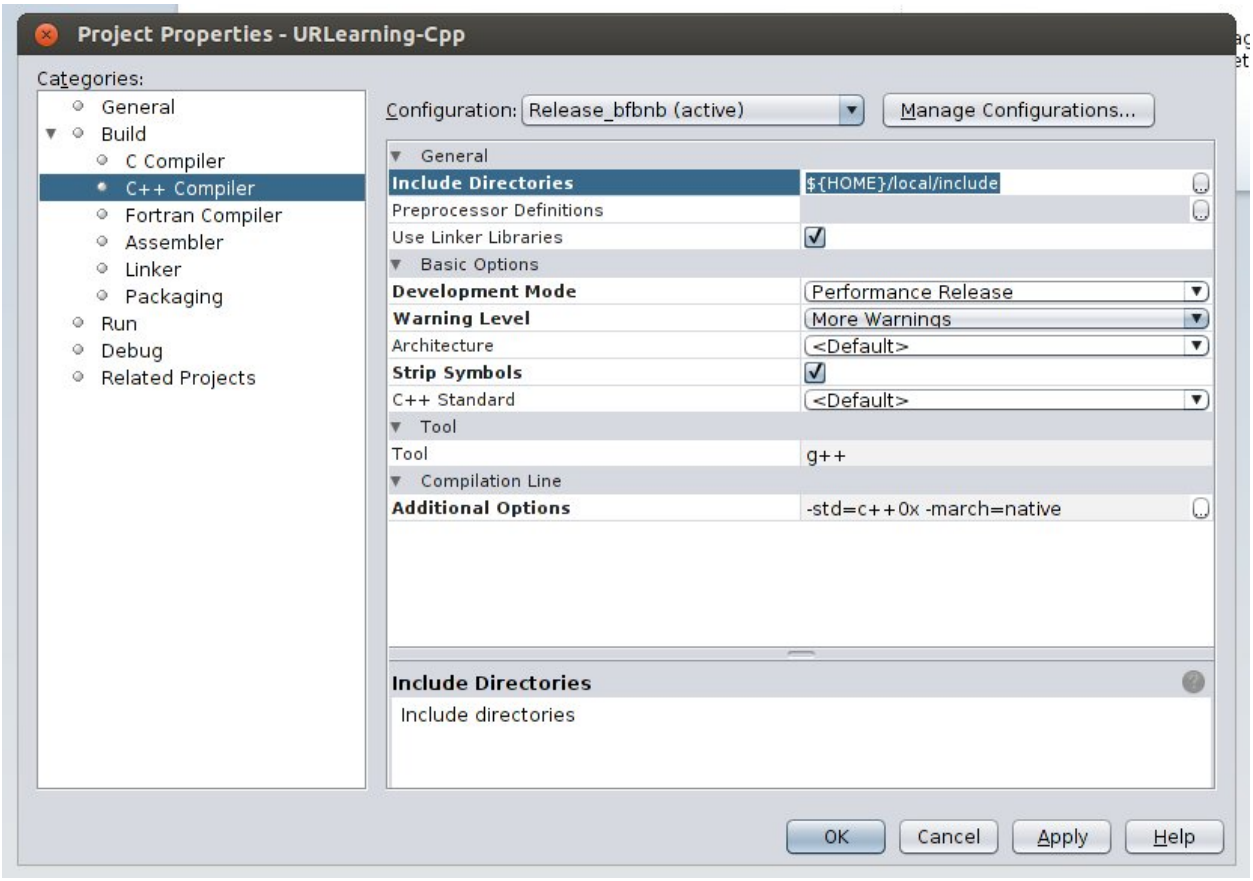


3. On the ‘C Compiler’ tab, update the ‘Include Directories’ to include the path to the boost header files. For example, on my machine, I have the directory $HOME/local/include/boost, so the Include Directories is ‘${HOME}/local/include’. (This step is probably not strictly necessary because boost is C++.)

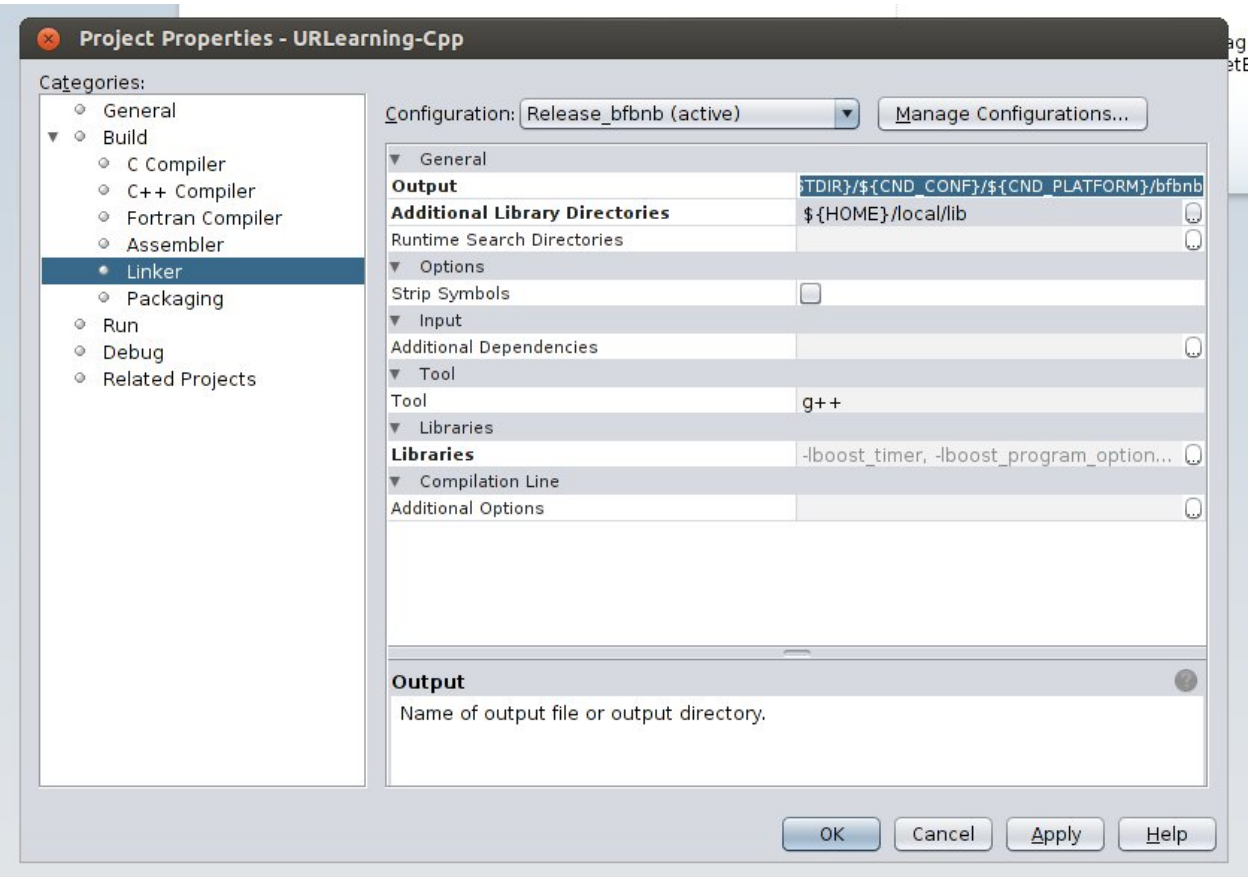
   All of the code is written on the assumption that the folder containing the boost headers is named “boost” (e.g., not “boost-1.54.0”, etc.). If the folder is named something different, then all of the include statements in the code will need to be updated accordingly.



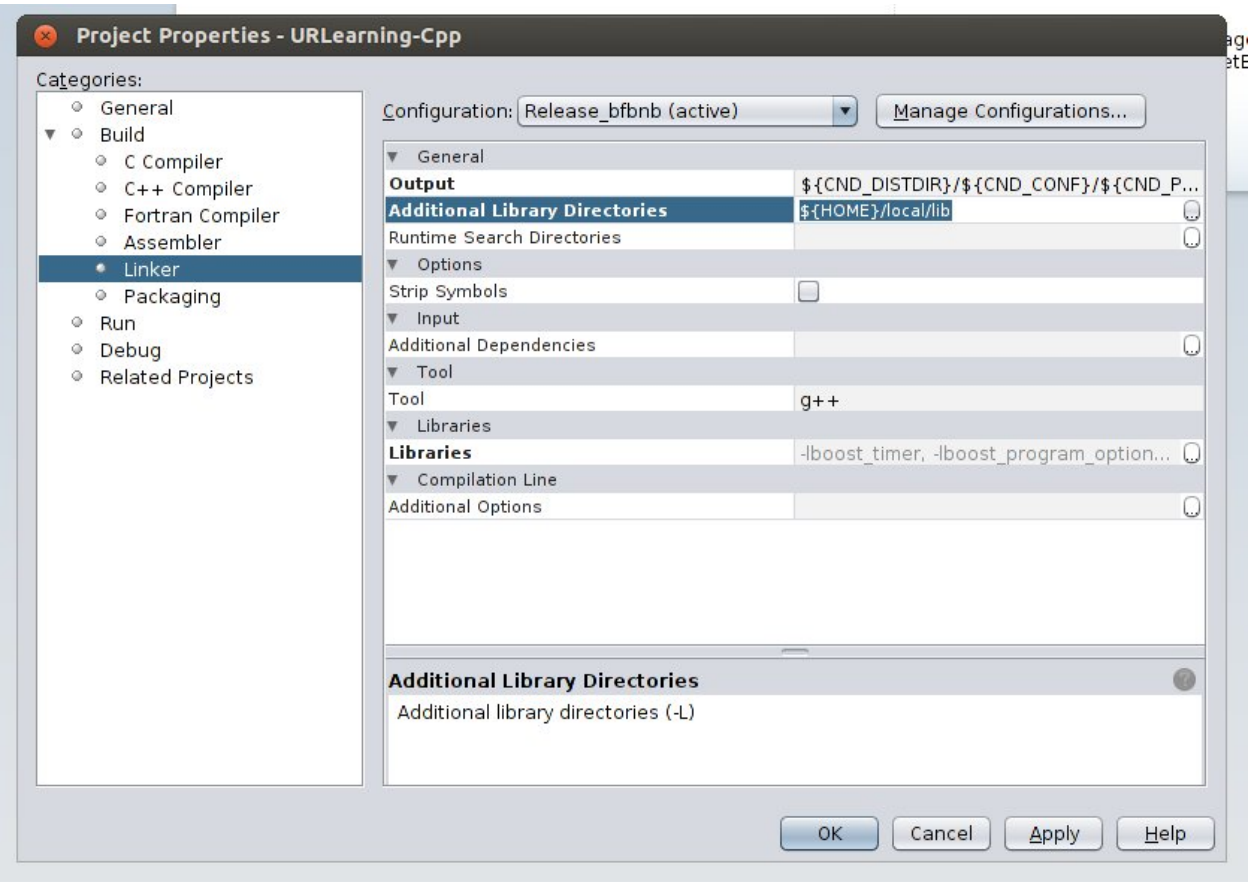
4. Also update the ‘Include Directories’ on the C++ tab.



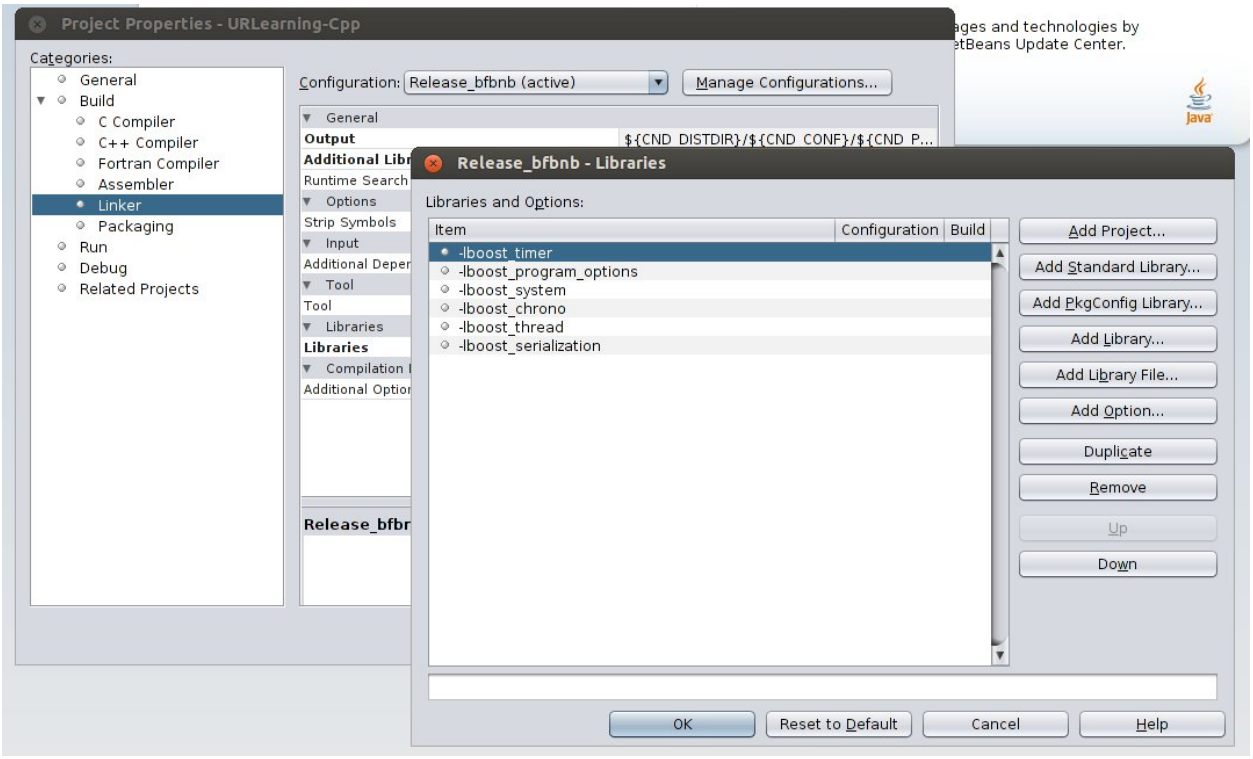
5. Set the name of the created executable. This is the ‘Output’ field on the ‘Linker’ tab. It should generally be something like: “${CND_DISTDIR}/${CND_CONF}/${CND_PLATFORM}/bfbnb”. Only the ‘bfbnb’ part should change.



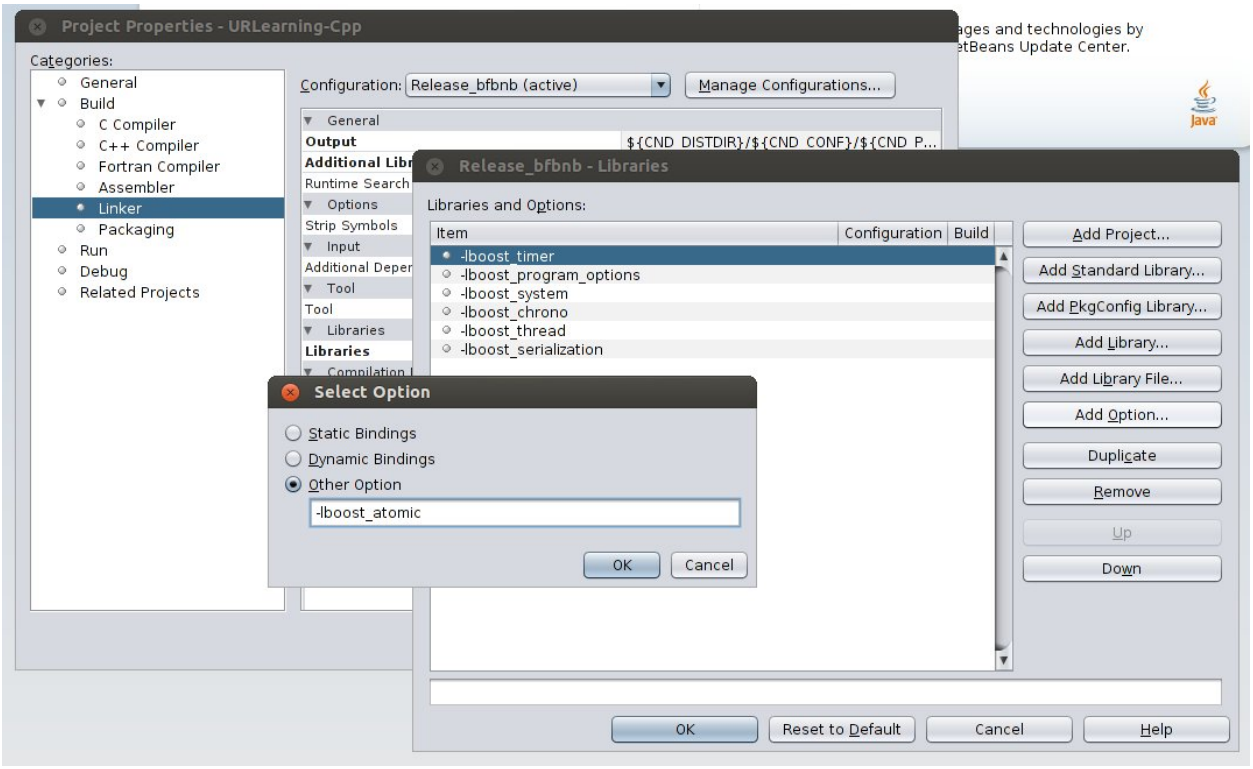
6. Add the path to the boost static libraries to the linker. For example, on my machine, I have the all of the boost static libraries in the directory $HOME/local/lib, so the 'Additional Library Directories' is '${HOME}/local/lib'.



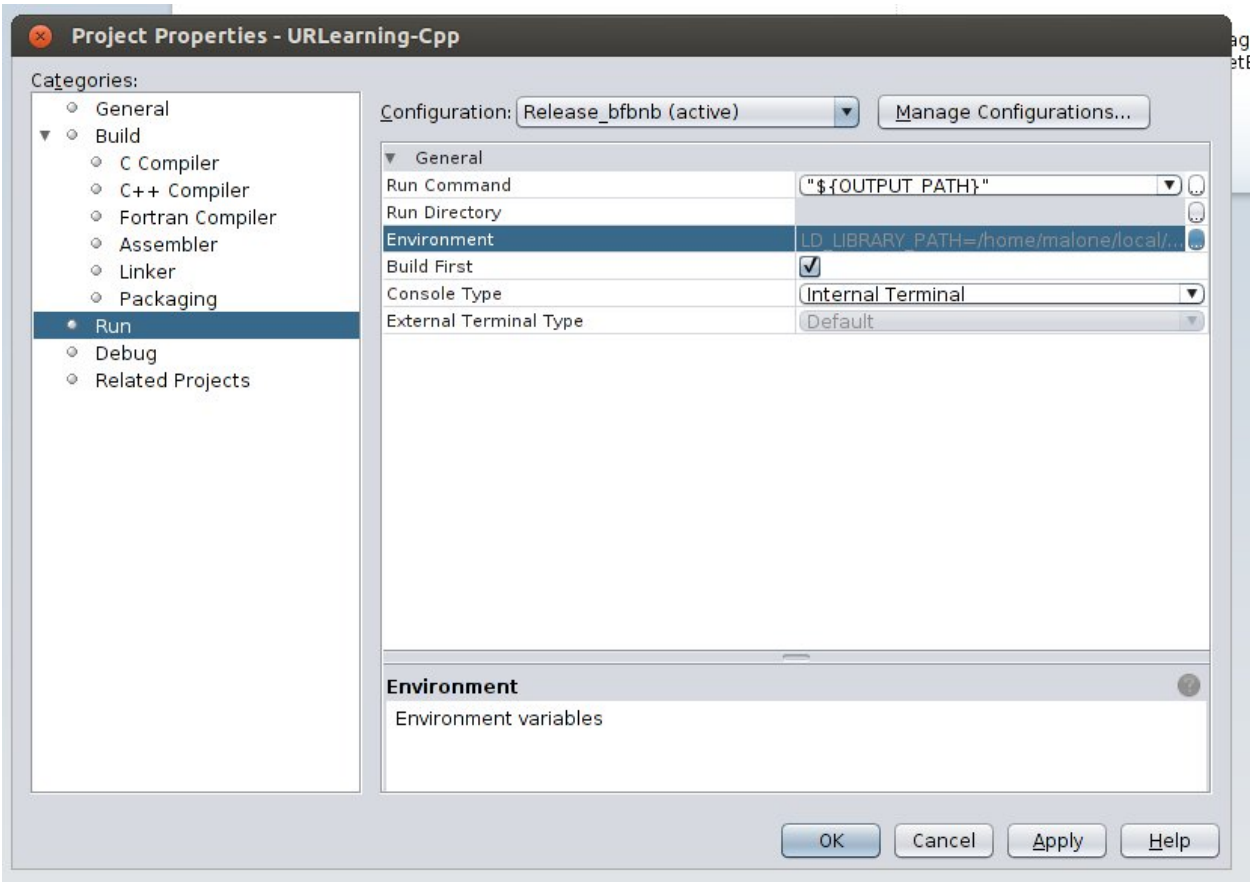
7. Add the necessary libraries to the linker. Click ‘Libraries’ on the ‘Linker tab and a popup window opens. (All of the necessary libraries should already be there for existing configurations.)



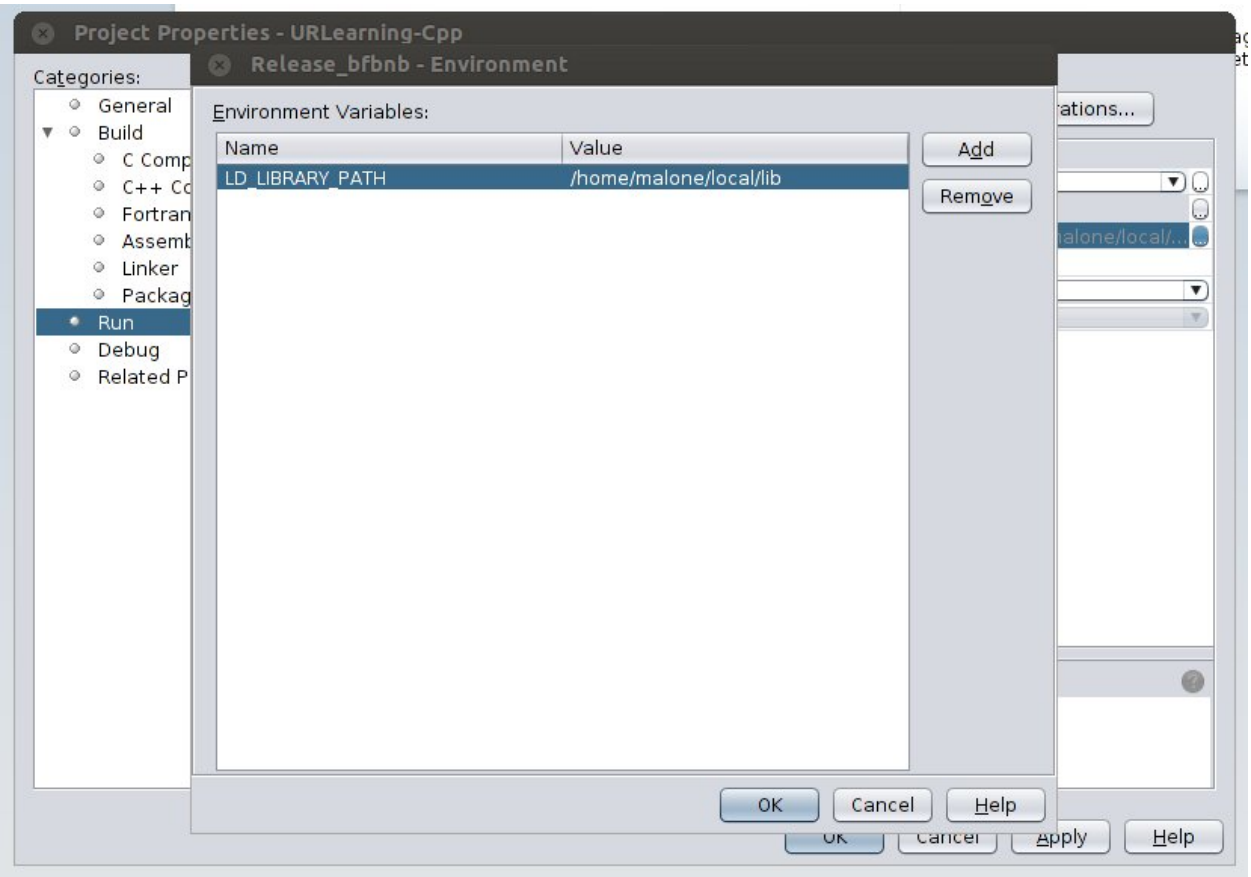
8. You can add new libraries by clicking ‘Add Option...’. A new dialog box will open, and the desired library can be added using ‘Other Option’ and typing the name into the text box.



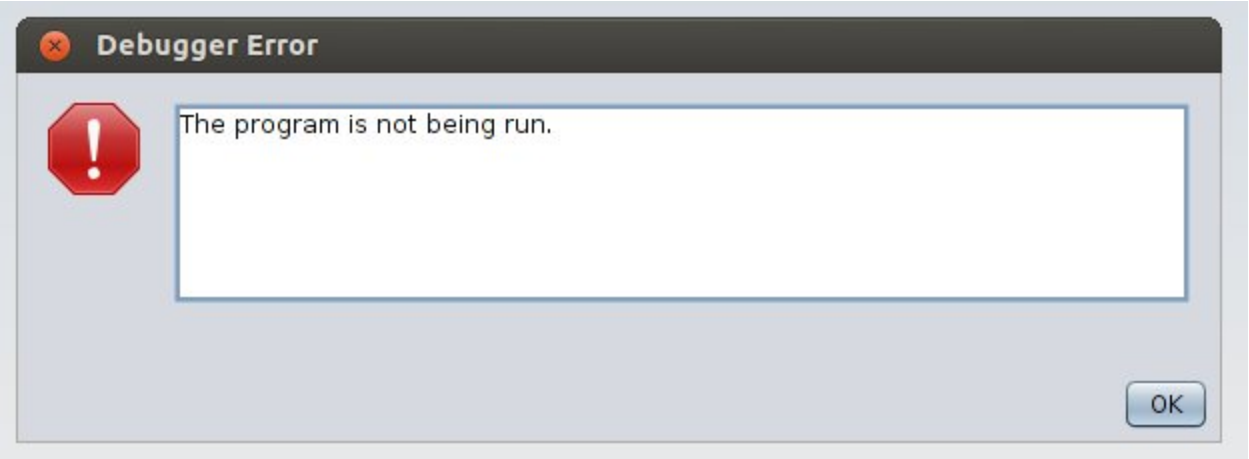
9. Add the path to the boost shared libraries by setting the ‘LD_LIBRARY_PATH’ environment variable. Click on ‘Environment’ on the ‘Run’ tab.



10. A popup window will open where you can either add or update the LD_LIBRARY_PATH variable to point to the directory with the shared boost libraries. For example, on my machine, I have the all of the boost shared libraries in the directory $HOME/local/lib, so the value' is '/home/malone/local/lib'. (I have not figured out how to use a macro for this variable. Neither '${HOME} nor '~' seem to work.)



11. This should be all of the configuration necessary within NetBeans. If you receive compile-time errors related to boost, make sure that the ‘Include Directories’ paths are correct. If you receive linker errors, then make sure the ‘Additional Libraries Directories’ paths are correct and that the ‘Libraries’ on the ‘Linker’ tab include all necessary libraries. If you receive a runtime error (see image below), then check that the LD_LIBRARY_PATH variable is correct.



There are multiple ways to tell NetBeans about the necessary boost libraries; however, I found this approach to be the simplest and most light-weight (e.g., this works for boost installations in your home directory, which makes managing versions easier).

12. The ‘buildAll.sh’ and ‘cleanAll.sh’ scripts can be used to compile the programs from the command line. This is necessary when compiling on a server and NetBeans cannot be used. (Again, there are other ways to accomplish this, but I found this to be a light-weight way to accomplish it.)

    These scripts just use the automatically generated makefiles; however, for them to work properly, the relevant environment variables have to be set. For example, I have the following lines in my .bashrc file (again, there are other ways to do this).

    ```
    export C_INCLUDE_PATH=$HOME/local/include:$C_INCLUDE_PATH
    export CPLUS_INCLUDE_PATH=$HOME/local/include:$CPLUS_INCLUDE_PATH
    export LD_LIBRARY_PATH=$HOME/local/lib:$LD_LIBRARY_PATH
    export LIBRARY_PATH=$LD_LIBRARY_PATH:$LIBRARY_PATH
    ```

    Also, for new configurations to be built (or cleaned), the relevant lines must be added to the scripts; they do not automatically detect anything. In addition to making the configurations, I also add a symlink to $HOME/local/bin for each of the programs.