CMake and its Application to the uMachine Compiler

Aidan Bickford

I. Introduction

CMAKE is a build management system which enables easy configuration and compilation of large software projects. CMake was the result of a National Library of the Medicine funded project aimed at developing a easy way to deploy software across multiple platforms [1]. As Hoffman and Martin explain central to its design was the ability to perform the task of Unix make files, and be able to generate Windows Visual Studio files. This was key to enabling cross platform development and deployment.

II. How It Works

Makes role is to coordinate the native build environment. This allows it to hand off the compilation of the projects to the host machine to which the build is targeted. This is done my placing the a CMakelists.txt file in each directory . Each CMakeLists file keeps track of the build files in that directly and any other external dependencies.

There are two different steps to the CMake build process [1]. The first is the configure step, followed by the generation step. The configuration step involves gathering all of the files and variables defined in the main CMakeLists.txt file and all of it's referenced subsidiary CMakeLists.txt files. The allows for the creation of a in memory model of the project that is going to be built. The generation step uses this in memory model to generate the code by compiling each of the targets or objects to be compiled. Dependancies between libraries or objects in the CMake build are handled by CMake only when there is no alternative, this means that for the most part dependancies are handled by the target IDE after the project has been passed off for generation.

III. Team Use

This was chosen for this project because it allows for the simple quick out of source builds across multiple platforms. This enabled the team to build to have multiple build configurations for the different platforms that were used between the students. The CMake configurations were create to target different IDE development environments. Eclipse and XCode were used on Mac, in addition to Visual Studio on Windows. A build configuration was also created for a Linux version of the compiler, this targeted the student test machine. This allowed for the simple creation of an executable for the evaluation and demonstration of the groups progress.

CMake was configured to build projects for these IDEs in externally contained source folders. This made it easy to separate project and IDE specific files from the project source code. This made organizing and creating commits strait forward, as it helped prevent the extraneous inclusion of build specific files.

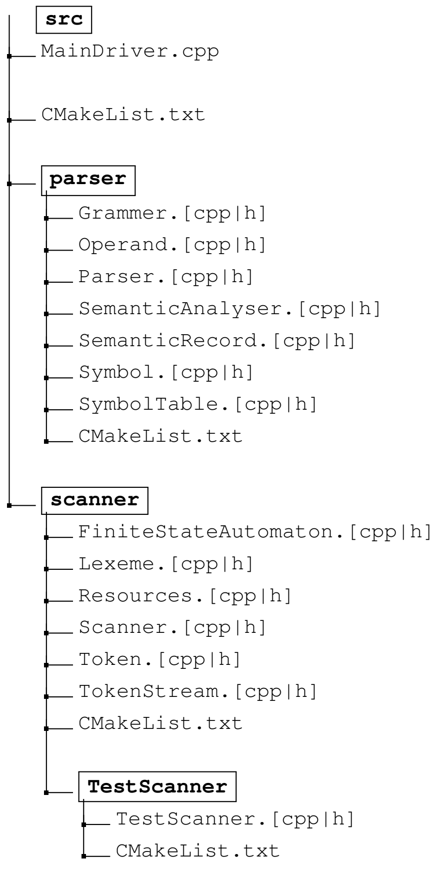

Fig. 1. Folder Structure

IV. Application to the compiler

Figure 1 shows the internal file structure for the compiler project. There is a CMake file for each of the different directories to keep track of all of the files and their dependancies. There are two different build targets that can be specified. The first drives the scanner unit tests from TestScanner[.h—cpp]. The second builds the the complier which is outlined below.

Figure 2 shows the CMakeLists file from the root directory of the compiler source directory. The third line indicates the name for the project will be ”app_project”, this will show up in IDE specific build which are created for a specific platform such as Visual Studio. The fifth and sixth lines add the subdirectories for the parser and the scanner, and the last two lines allow for the sub-files to be included in the

```
1  cmake_minimum_required(VERSION 3.0)
2
3  project(app_project)
4
5  add_subdirectory(scanner)
6  add_subdirectory(parser)
7
8  add_executable( mp MainDriver.cpp)
9
10 target_link_libraries(mp mpparser)
11 target_link_libraries(mp mpscanner)
12
13 install(TARGETS mp DESTINATION bin)
14
15 include_directories(${CMAKE_CURRENT_SOURCE_DIR}/scanner/)
16 include_directories(${CMAKE_CURRENT_SOURCE_DIR}/parser/)
```

Fig. 2. Root Level CMake

MainDriver.cpp which is the only file in this directory. These directories hold the statically linked libraries for the creation and the scanner and the parser, which are set to link against on the tenth and eleventh line. The eight line creates the target ”mp” which will be the name of the compiler application generated.

V. Conclusion

CMake is a good solution for building and deploying projects on multiple platforms. It offers integration with several of the more common IDEs such as VisualStudio and Eclipse. This feature was instrumental in allowing the team to collaborate on the construction of the uMachine.

References

[1] B. Hoffman and K. Martin *CMake*, AOSABook.org, http://www.aosabook.org/en/cmake.html