CodeMate

An automate tool for programming
and templating

by Li Dong (dongli@lasg.iap.ac.cn)

1 Introduction

CodeMate is an automate tool for processing and building project (currently only in Fortran[1]). It is currently in alpha testing stage, any feedback is welcome.

1.1 Why CodeMate?

There are many programming languages for helping people to solve problems, and generally they fall into two categories, one is compiling languages and the other is interpreted languages. The major target language that CodeMate deals with is Fortran, which is the first high-level language and is mainly used in numerical computation due to its high efficiency.

(some more background?)

It is tedious and error-prone to manage the Makefile of a project. Sometimes, user just wants to write a small and quick project for proving concept, but writing the dependencies among the codes and linking with external libraries is such a headache. In the other hand, even a number of starting simple codes maybe group up into a huge project with hundreds of codes. In this case, the headache turns to be serious.

For most other languages, like C++ and Java, there are many great integrated development environment (aka IDE) to do the dirty jobs. Fortran only gets old Compaq Visual Fortran, expensive Intel Visual Fortran, and several less used IDEs. Almost all of them are only available in Windows, but most serious Fortran developers work in Linux or Mac. So CodeMate comes into business to close the gap for Fortran users (most of them are scientific programmers), and provides more capabilities that IDEs do not have. Also CodeMate is lightweight compared with IDEs, it is just one command at all (that is `codemate`), so it can be run on the remote server with only command line interface.

Using CodeMate, user will be liberated from writing Makefile or other duplicate stuffs. No file editing is needed for basic uses. In addition, CodeMate will do more thing automatically, such as linking external libraries automatically, and processing templates.

1.2 CodeMate Capabilities

CodeMate can scan a Fortran project to extract internal and external dependencies, and create a Makefile for it. For external dependencies, CodeMate has some predefined library profiles, which can be expanded. When a library is used (e.g. `use netcdf`), CodeMate will automatically write a Makefile that knows to link with that library. So users are free from worrying about the manual linking stuffs, once the environment is set up correctly. The above automation capabilities may seem to be plain (but it is really handy), the outstanding feature of CodeMate is adding template

[1] Only Fortran 90 and later standards are supported, since we are in 21st century!

mechanism to Fortran. If there are any template instance and CodeMate knows about the template definition, the template instance will be processed to generate a full code internally. This can really save a lot of typings and make the program tighter and more manageable. After these operations, user can just invoke `make` to build the project.

1.3 Template

Template, or more precisely template metaprogramming, is a technique to transform the original codes into temporary codes based on some rules. Many morden programming languages support it natively (by compiler), such as C++. Unfortunately, Fortran does not support it formally. That means when you want to use some advanced data structure like linked list, you have to reinvent the wheels over and over again, due to the limited generic programming and the safe but also limited pointer. Fortran has its reason to insist these limitations, that is to ease the compiler optimization, but in real and modern applications, it is difficult and less maintainable to just use the basic data structure. CodeMate tries to solve this problem by constructing a practical template framework, which uses a customized Fortran parser[2], interface and dynamic compilation mechanism of Java. Do not be scared by Java, since normal users will not contact with it directly. Only advanced users who want to write templates need to learn some basic Java syntax. Once familiar with it, you will find out that you can create any truly useful templates with the full power of Java.

[2] The customized Fortran parser is generated by using ANTLR.

2 Tutorial

2.1 Installation

Since CodeMate is lightweight, installing it should be very easy. First download the installer binary from CodeMate repository as in shell by using `wget`:

```
$ wget https://github.com/dongli/CodeMate/raw/master/products/installer/codemate.installer
```

Then run the installer as:

```
$ chmod a+x codemate.installer

$ ./codemate.installer
```

CodeMate will be installed in `$HOME/.codemate`, and a `source` statement will be appended to the configuration file of the shell (.bashrc for BASH).

After the installation and login to the terminal again, `codemate` command will be available. Command line autocompletion is also added for BASH, so enjoy the TAB.

CodeMate can also be updated after installation by invoking

```
$ codemate update
```

It will check the status of CodeMate in the remote repository. If newer one is available, it will replace the older one.

2.1 Configuration

(Configuration is simple)

2.2 Basic Usage

CodeMate tries to minimize the workflow of building a project, so it does. Only one operator (or subcommand) is needed normally, that is `scan`:

```
$ codemate scan <project root>/<single code>
```

When a directory is provided, CodeMate will consider it as the root of the project, and any Fortran code (with suffix as .F90 or .f90) will be processed. When only a single code is provided, the code will be parsed and print the rewritten code onto the console.

2.3 Template Usage

Template is composed of three parts: 1) template name; 2) template arguments; 3) template block, as the following example:

```
foo<a, b, c> { ... }
```

where `foo` is the template name, `a, b, c` are the template arguments and the code inside curly braces is the template block. It is worth noting that the template block is the extra weapon that CodeMate provides to deal with the real and tough problems. By using it, users can input any Fortran code to the template processor in CodeMate, then the processor can customize the code based on the template definition and arguments. It is similar to the block concept in Ruby, so the resulting code can be tweaked much more flexibly. Actually, the block is some kind of argument for the template, but it is code.

There can be zero argument, but the angle brackets must be reserved to indicate the template instance. Whereas the curly braces should be omitted, when template block is absent.

CodeMate provides some built-in templates that can be used directly, as listed in Appendix A.

(Add some examples)

3 Advanced Topics

3.1 Write a Compiler Mate

Setting compiler and its options is one of the most headache things when building a project. Most modern IDEs support easy switching among different compilers, and let developer choose which compiling scheme (Debug or Release) to use. In scientific programming world, the general practice is to write down the options for each compiler in Makefile, and judge which option set to use based on the compiler. This will make Makefile hard to read, and less maintainable. In addition, there is no common labor-saving solution for this problem.

CodeMate addresses this problem by introducing an interface for specifying compilers. Once a compiler is specified in a Java class that implements this interface, all other users can benefit from it, so it is *one thing for all*. User only needs to tell which compiler to use in the configuration file, then CodeMate will invoke the correct compiler to do the building job[3].

class method	
getVendorName	This method returns the name of the vendor, such as “Intel” and “GNU”.
getCommandName(String language)	This method returns the name of the compiler command for the given language, such as “ifort” and “gfortran”.
getDebugOptions	This method returns the options for debugging.

[3] This is actually done by Make.

getReleaseOptions	This method returns the options for releasing including several optimization options.
isAvailable	This method should check if the compiler is available in the system. If yes, return true, otherwise return false.

3.2 Write a Library Mate

Linking with external libraries is also a boring job. User not only needs to specify the location of a library, but also the linking options that are difficult to remember.

3.3 Write a Template

Appendixes

A Built-in Templates

Templates are used to save the developing time of users. In CodeMate, there are some built-in templates ready for using. If you have some good templates, sharing them with all the users would be a good way to realize their full value. For the time being, contact author for sharing details please.

Bundle Name	Template Names
`FortranListTemplate`	`list_elem_t<T>, list_t<T>`

B Built-in Compiler Mates

C Built-in Library Mates

D To-do List

1. Add sanity checking for compilers, libraries.
2. Add an operator that user can use to submit good templates, mates of compiler or library.