COMS4115 Final Project Report
Text Manipulation Language (TML)

Austin Lee
Al2570
Summer 2008
taehee@gmail.com

Table of Contents

1. Introduction

Regular Expressions have a wide array of applications and are frequently used in software development, especially in text processing. Development tools such as Perl and vi have built-in support for regular expressions and experienced users of Perl may find picking up regular expressions an easy task. However, in many cases, developers need a fast and easy way to process large amounts of text data and do not have the time to learn Perl and/or the unique syntax of regular expressions. To bridge this gap, I decided to design a simple, easy-to-use language that people can quickly learn and be able to use to search and/or replace their text data.

The primary objective I wish to accomplish with Text Manipulation Language is to bridge the aforementioned gap by making the syntax of the language simple and easy to remember. The emphasis in designing the grammar was on using words rather than symbols such as "~", "*", "^" etc. From my own experience, the lack of intuitiveness in the selection of these symbols made it difficult for me to remember which one meant what and, unless I used them very frequently, it took time for me to re-learn the symbols every time I needed to use regular expressions.

My intention was not to rewrite or redesign regular expressions, but to find ways to augment them and facilitate user learning. Hence, I focused on syntactical improvement and proper handling of the language using ANTLR (v2). The actual processing/execution of regular expression search and replacement is provided by an open-source library called Boost.Regex. Boost is one of the largest open-source developer communities for C++ libraries. Also, time constraint meant that I needed to limit the scope of TML to maximize the usefulness of the existing regular expressions while introducing something new and valuable to other users. Therefore, many aspects of regular expressions remain unchanged in TML. For instance, the syntax used for character classes and string replacement remains the same (passed directly to the Boost regular expression engine without modification by TML).

2. Language Tutorial

2.1 Pattern

TML defines a pattern as either a string literal or a concatenation of sub-patterns. A string literal can be any sequence of literals enclosed by double quotes. A sub-pattern is either a composite pattern or a string literal or a group consisting of any of the previous two. Lastly, a composite pattern is a sequence of character expressions optionally coupled with quantifiers that specify the occurrence pattern.

Let's review TML's pattern specification one more time. A pattern is either a string ("hello world") or a concatenation of sub-patterns where each sub-pattern is enclosed by an open parenthesis and a close parenthesis:

```
p = ( sub-pattern-1 ) ( sub-pattern-2 ) … ( sub-pattern-n)
```

Here, any of the sub-patterns can be an identifier referencing an already declared and defined pattern before the definition of p.

A character expression can be either a character class expressed in the form as the standard regular expressions, e.g. [a-zA-Z], or a keyword such as "word" (\w in Perl) and "digit" (\d). A sub-pattern can look something like this:

```
p1 = ([yY][eE][sS] digit digit word nonword)
```

The quantifiers apply to the string or character expression directly to its left and have the following format in TML:

```
p2 = (digit 2-3)
```

The above pattern will tell TML to look for a number consisting of two or three digits.

2.2 Search

Users of TML can define as many searches as they want. Each search must specify the input to be searched and a pattern to look for in the input. If no action is specified, the default behavior is to find only. If a user wants to replace strings matching the pattern with something else, he/she must set the search action to "replace" and specify the replace string. Here is an example search definition:

```
mySearch.s_input = file1;
mySearch.s_action = replace;
mySearch.s_pattern = p;
mySearch.s_replace = "$2$1";  //swap the first and second sub-patterns
```

To execute this search, you specify the following:

regexp_engine(mySearch);

The search execute command can be used again and again. Any search can be re-defined or modified and passed to the search engine.

2.3 Input/Output files

TML allows two ways of specifying input files. An input file can be either a single file (of type "file") or a file that contains a list of input files (of type "file_list"). If a file list is specified to a search, the pattern for this search will apply the same pattern to each file repeatedly. This will achieve the same effect as passing each file to the search one at a time.

In both cases, TML generates a result file for each input file it processes. The name of the result file is the name of the input plus the ".tmp" extension.

If the search action is find-only, then the result file will contain the results of pattern matches with line numbers and position in the line where each match occurred. It also shows the string that was matched (blank for space or white space). Here is an example of what a result file may look like:

Line 1: (pos: 1) word
Line 4: (pos: 9) word

For the case of replace, the resulting .tmp file will contain the input file text with appropriate parts of the text replaced with the user-supplied replace string/pattern.

2.4 TML file

Here is an example .tml file:

```
pattern p1, p2, p3, p4;
p1 = "hello";
p2 = (space 0-1);
p3 = ("world" optional);
p4 = (p1)(p2)(p3);
file f = "input.txt";
//this is just a single line comment
search s;
s.s_input = f;
s.s_pattern = p4;
s.s_action = replace;
s.s_replace = "foo";
regexp_engine(s);
```

3. Language Reference Manual

3.1 Lexical conventions

3.1.1 comments

TML uses the widely used single-line comment style used by many languages. I.e. any string starting with "//" up to the end-of-line character is considered a comment. For simplicity, I did not include support for multi-line comments such as /* … */, but this can be easily added to a later version of TML.

3.1.2 Identifiers

TML uses the conventional syntax for its identifiers, i.e. any string that starts with a letter or an underscore ("_") followed by zero or more letters or digits or underscores.

3.1.3 Keywords

In addition to the keywords reserved for types (see 3.2), the following keywords are used to represent certain character expressions used by regular expressions.

TML	RE
word	\w
nonword	\W
digit	\d
nondigit	\D
dot	.
tab	\t
eol	\n
space	\s
ws	\s*
optional	?

Furthermore, the following keywords have been introduced to augment the RE syntax:

TML	meaning
beginsWith	"^" in RE, must match from the beginning of line
bw	same as "beginsWith", made short for simplicty
endsWith	"$" in RE, end of pattern must match from the end of the line
ew	same as "endsWith", made short for simplicity
except	[^ …], used inside character class in RE to specify class of characters to be ignored
greedy	"*" in RE, match as much as possible

3.1.4 Strings

A string or a string literal in TML, which is written in C++, follows the definition of strings in C++ and therefore follows the same rules for escaping characters such as a double quote within the string.

Any character sequence that must be recognized as a string (literal) must be enclosed by double quotes.

3.1.5 Other tokens

The following tokens are also used in TML:

(	)	-	;	[	]
=	.	$	‘	|	

3.2 Types

The classes that represent patterns, searches and files are all derived from the same parent class called MyBasicVar whose only private member variable simply contains the name of the variable which is one thing all instances of these classes will have in common. This allowed inheritance of the same get/setVarName() functions which reduced coding efforts by reuse of the parent’s functions.

3.2.1 Pattern

The Pattern class describes a pattern or a regular expression. Two notable members of this class are the string that represents the RE in the form acceptable by Boost.Regex and a vector of Patterns for the case in which the pattern is a grouping of multiple sub-patterns.

3.2.2 Search

The Search class represents data and actions necessary to perform a search of a pattern in one or more input files. The Search header file (search.h) shows what settings are set to what values by default. For example, each search is case-sensitive by default unless otherwise specified by the user.

3.2.3 File/File_list

The InputFile class represents a file or a list of files to be searched. Due to its simplicity, it will be part of tml_common_defs.h along with other common typedefs.

3.3 Statement/Expression

Every statement in TML must end with a semicolon which is the separator/delimiter of statements. Each statement is an expression which can be a variable declaration, a variable definition or a search execute command.

3.3.1 Declaration

A variable of any type is declared in the C/C++ manner. All variables of all types are declared the same way, i.e. the type name followed by whitespace and an identifier or a comma-separated list of identifiers. TML does not yet support definition and declaration in a single statement, i.e pattern p = “foo”; is not allowed.

3.3.2 Definition

A string literal pattern and a file/file list are defined the same way—the identifier on the left and the string on the right. Defining composite patterns are slightly complicated to be able to represent arbitrary text sequence. Search definition is a multi-step process, but is fairly straightforward.

3.3.3 Search defintion

Here are the search options.

s_input – the identifier for the file input
s_action – none for search only and “replace” for search and replace action
s_mode – “case” for case-sensitive search, “nocase” for case-insensitive search
“beginsWith” or “bw” to prefix with “^” for matching from the beginning of the line
“endsWith” or “ew” to suffix with “$” for matching from the end of the line
“reset” which corresponds to “\G” option in Perl, makes the pattern search reset and continue after the first successful match
s_pattern – the identifier for the pattern to be used
s_replace – the replace string, follows the Perl convention (as does Boost.Regex)

3.3.4 Pattern definition

A composite pattern can consist of any combination of the following.

3.3.4.1 Character class

A character class in TML takes exactly the same form that Perl uses with one exception. To maintain the goal of this language, TML uses the keyword “except” to represent the “^” symbol used inside the square brackets to negate the character class in RE. For instance, [except y] will match any character except “y”.

3.3.4.2 Quantifiers

I found the quantifiers used by Perl or any other tool that implements regular expressions a bit confusing so I tried to make them simpler in TML.

TML uses simple from-to syntax for this purpose. For example, "0-2" means zeor, one or two times. "-9" means zero to nine times (equivalent to {0,9} in RE).

Also, the keyword "optional" equates to the optional symbol "?" in RE.

In Perl, "?" is also used to make a pattern non-greedy, meaning it will consume as few characters as possible to match the pattern. "\d*?" is an example of non-greedy. In order to achieve the same effect in TML, you can simply use "(\d -1)" meaning it will stop as soon as it matches up to one digit. Conversely, "0-" means zero or more times which equates to "*" in RE.

3.3.4.3 Grouping

TML uses the same simple syntax for grouping as Perl which is to "OR" patterns using a single vertical bar "|". A grouping of any pattern identifiers or composite patterns makes the pattern also a composite pattern. A member of a group can be either an identifier that references another pattern or a composite pattern which itself is not a group.

3.3.5 Search engine

The regexp_engine() statement invoked on the search passed to it as the single function parameter, performs search on the input file using the user-specified pattern. This statement encapsulates the action of performing the search as opposed to declaring or defining.

3.4 Boost.Regex

In order to handle the actual pattern searching and/or text replacement, I am using a Boost library called regex. This library is known to be highly portable and reliable and has almost complete support for RE. Only the "search" and "replace" functions of this library were used, but further improvements to TML may require additional functionality support from Boost.Regex.

The current TML program has been developed on Mac OS X version 10.5.4 using the GNU g++ compiler. The necessary Boost.Regex library has also been built and installed from the source downloaded from boost.org. Anyone wishing to use TML on other platforms (it was developed and tested on Windows XP and Open SuSE 10 during early stages of the development only) must also build and have available the Boost regex library compatible with his/her build system.

3.5 Error reporting

The Logger class is a singleton class which was designed to provide flexibility in both reporting errors and printing out debugging statements by a number of logging levels. A single instance of this class used throughout the program. Since TML is currently a single-threaded program, this module was not designed for multi-threaded environments. I.e. it assumes its logging function is accessed one at a time.

The logger allows debugging statements to be printed in product mode by setting the log level to 1. The level log of 0 means the debugging statements are printed only when DEBUG is defined.

Errors are largely classified as being file I/O related, syntax related and RE search related.

4. Project Plan

4.1 Project Process

The process I followed for developing TML was a very straightforward, simplified version of the waterfall. I finished the design first and then coded and unit tested and the test design and unit test overlapped slightly and finished up with function testing. No design or test design documents were written (except for this report).

4.2 Programming Style Guide

4.2.1 Antlr

For grammar rules, I put the rule name on one line and its terminals and non-terminals on the following lines all lined up vertically with proper tabbing.

```
rule
        : non-terminal
        | terminal
        ;
```

For the AST tree walker, if it is a short one-liner, I put the code directly next to the matching rule, otherwise, I put it below the rule indented properly to easily see which code corresponds to which rule.

```
rule
        : (TOKEN rule { foo(); }
or

rule
        : (TOKEN rule
                        {
                                some_long_function();
                                more();
                                …
                        }
```

4.2.2 C++

In general, I used thisConvention for variable and function names and _thisConvention for private member variables.

One other thing I made consistent throughout is using this convention for blocks

```
funcName()
{
        {
                …
        }
}
```

over

```
funcName() {
        …
}
```

Also, for comments, I preferred the single line comments "//" to /* .. */ whenever possible.

4.3 Project Timeline

Deadlines	Content	Comments
06/04/2008	Proposal	Submitted, received feedback from Prof. Edwards via e-mail
06/20/2008	LRM	Late submission, language design needs much re-work
08/11/2008	Final report	On track

4.4 Development Environment

For this project, I chose to use my Macbook Pro which comes with Xcode 3.0. This was the first time using this IDE for heavy programming.

Xcode was primarily used for text editing and debug building. I used Xcode to build debug versions of Antlr and Boost as well as TML. All of the source for TML, Antlr and Boost was bundled together as a single project (only for debug).

I created a Makefile to produce a release version of TML. This is the version tested and prepared for project turn-in. I needed to also generate antlr and boost libraries from source using the configure and make facilities that came with the downloaded source. All header files and generated libraries were installed on my system and paths to them were specified in the Makefile.

4.4.1 Operating System

My initial goal was to port TML to at least two other operating systems—Windows XP and Linux. I set up my Mac as the primary development system, but I had also set up a Windows machine and a SuSE 10 machine to work with my project files. In the end, due to time constraint, development of TML continued on Mac Leopard v10.5.4, but additional testing on XP and Linux discontinued.

4.4.2 C++

My secondary objectives for this project was to become more familiarized with C++ standard libraries and the language in general. Therefore, I chose to do this project in C++. All of the code I wrote is in C++ as well as Antlr-generated code.

4.4.3 Antlr

Since the course materials contain examples using Antlr v2, I decided to use the same version of Antlr instead of the latest version 3. Antlr is a tool or a set of tools that are used to facility compiler or interpreter development. I used Antlr to build the front-end of TML.

4.5 Project log

June 23 – July 11
Finalize design and LRM

July 12 – 25
Write code, unit test

July 26 – August 4
Continue coding, more focus on final

August 5 – August 11
Function testing, document write-up

5. Architectural Design

5.1 Front-end

The front-end of TML consists of a lexer, a parser and an AST tree walker. The lexer for TML, TMLLexer, extends Antlr's Lexer to define all the tokens needed by the parser to tokenize the input stream. Tokenized input stream is then passed to TMLParser, which again extends Antlr's Parser to build an abstract syntax tree. Once the AST tree building is complete, the AST tree walker for TML, TMLTreeWalker, takes over and executes the appropriate actions. These actions represent interaction with the back-end to make sense out of the syntax tree. TMLLexer, TMLParser and TMLTreeWalker are all contained within the grammar.g file.

5.2 Back-end

The back-end of TML consists of a driver, a set of functions for managing statement declaration and definitions, a pattern module, a search module, and a logging facility.

The TMLTreeWalkerHelper module interacts directly with TMLTreeWalker to interpret expressions parsed by the front-end. It processes variable declarations by making sure each type is properly named, adding new variable declarations to the symbol table, type-checking and so forth. Its key responsibility is to properly handle variable definitions and declarations and populating the symbol table.

The symbol table for TML is implemented using a vector of maps where the key is the variable name and the value is the type of the variable. You can look up a variable to see if it has already been declared, check its type to ensure proper type association and add a new variable-type pair to the table. The symbol table is static and global, i.e. it is accessible to all other modules (in the back-end).

In addition to the symbol table, three additional global lists are used in TML. A template class is used to define a list handler type that works as a general container. This list handler is templatized so that it can work with pattern objects, search objects and inputfile objects. The patternListHandler, for example, keeps track of all patterns defined. A new pattern object is not created and added to the list handler until it is defined. Merely declaring a new pattern does not initiate an object instantiation. It only creates an entry in the symbol table. The same holds true for searches and inputfiles.

Pattern, search and inputfile modules provide data and functionality to maintain and utilize variables of respective types. Detailed information on these modules was provided in the LRM section.

Logging facility is provided by the Logger module/class. At this point, this logger is equipped with the ability to print messages of type char* or std::string. It also takes in an error code as the second function parameter that gets translated into a string that

corresponds to the error code. Further improvement of this facility is desirable to provide detailed analysis when problems arise during program execution.

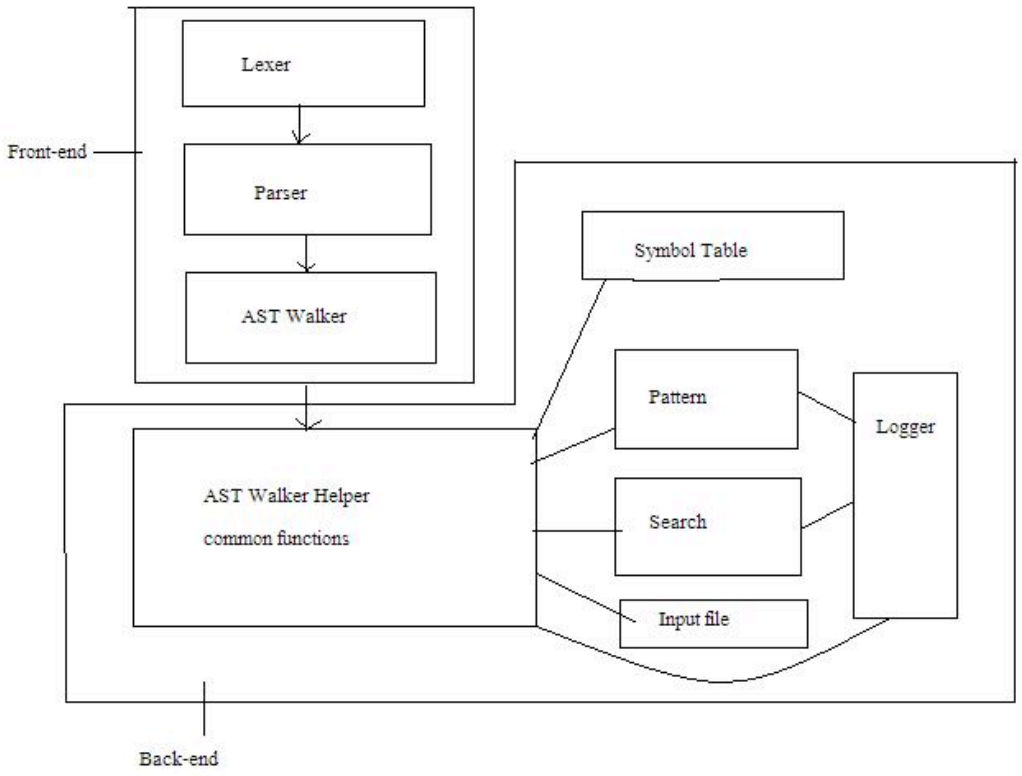
5.3 Data flow

The input .tmp file (the language file) is first read in to memory by the driver module (tmp.cpp) and it is then passed to the front-end. The data in memory flows through the front-end, going through the tokenizer and then the parser, resulting in an AST tree.

The AST tree is then traversed by the tree walker that calls back-end functions to interpret each node for meaningful information.

Once all pattern, file and search information has been extracted from the AST tree, a search execution on a search triggers the search object to call its performSearch function that reads the input file (the text data file), performs search actions using Boost.Regex and creates report files.

Once the search action completes, the AST walker returns to the driver and the program terminates.



6. Test Plan

6.1 Unit test

I built the entire Antlr, Boost.Regex and TML source within the Xcode IDE as a single C++ utility project. Xcode allowed me to build both release and debug versions of the program. Once I was able to set this up, I was able to step through every line of code inside TML, Antlr and Boost. This helped my unit testing efforts tremendously.

Unit testing was done incrementally as I added new code to the project. I would typically add a few lines of code, rebuild the project and use the debugger and breakpoints to step through each line of added code to make sure code paths were taken as intended and valid outputs generated.

The ability to step through Antlr-generated output was especially helpful during the debugging phase of the grammar and the AST tree walker.

6.2 Function test

So far I have created 11 function verification test cases. I created a test directory under the src directory and the test directory contains a subdirectory for each test case. Each test case consists of a tml file, an input file and an expected output file with the .out extension. The result of each test case is a .tmp file that gets "diff"ed against the pre-generated .out file.

The 11 test cases are broken into a file list test case (test case 0) and 10 test cases that use only input files of type "file." A file list is simply a collection of files and therefore orthogonal testing needed specifically for the file type function is minimal. This is why I made this test case the first one (indexed zero) so that additional test cases can be added with indexes above 10.

Makefile contains a target called "test" and "make test" copies all test files out of the src directory into its own "test" directory under the build root.

The test directory comes with a shell script called "runtest.sh" which takes in a single argument that equates the last index of the test case (the test cases are zero-based so right now the script is invoked with the first argument equal to 10). The script generates a result file for each test run and the result file contains the result of "diff" between the .out file and the .tmp file. If diff returns null, then the test case passed. Otherwise, it failed. At the end of the test run, the script runs "grep" on all result files producing an output to console displaying overall test status. Here is an example output:

```
Running TML regression test suites
Running TC1 ...
Running TC2 ...
Running TC3 ...
```

```
Running TC4 ...
Running TC5 ...
Running TC6 ...
Running TC7 ...
Running TC8 ...
Running TC9 ...
Running TC10 ...
Running the default file_list TC...
Tests completed
Test Results
result0.txt:1 passed
result0.txt:2 passed
result0.txt:3 passed
result0.txt:4 passed
result0.txt:5 passed
result0.txt:1 passed
result0.txt:2 passed
result0.txt:3 passed
result0.txt:4 passed
result0.txt:5 passed
result1.txt:Passed
result10.txt:Passed
result2.txt:Passed
result3.txt:Passed
result4.txt:Passed
result5.txt:Passed
result6.txt:Passed
result7.txt:Passed
result8.txt:Passed
result9.txt:Passed
```

Theses test cases have been useful as regression tests; however, they are by no means complete. Additional function level and even integration level testing (system tests, memory leaks, etc) will certainly improve overall quality of TML.

6.3 Test cases

Here is an example of a test case.

tc5/re5.tml:

```
//Simple find_only case
//Using a composite of multiple patterns
//A slightly more complicated case
pattern p1,p2,p3;
p1 = "world";
```

```
p2 = ("hello" optional);
p3 = ("w" 0-)("ord" 1-);
file f1;
f1 = "tc5/input.txt";
search s1;
s1.s_input = f1;
s1.s_pattern = p3;
regexp_engine(s1);
```

Here is the input data file.

tc5/input.txt:

```
word 123 word 123 wod wrd
123 word 123 hello 23 word
ord 123 world 123
word 123 ord 123
rd 123 ord 123 hello
wrd 123 wod 123 hello 234 world 312 word
```

7. Lessons Learned

7.1 Designing a language is hard

It certainly did not help that the summer semesters are much shorter. I found myself constantly re-designing and throwing out ideas that initially seemed sound. My biggest mistake was spending too much time to get the language design finalized. And the main reason it took so long was because, as I learned more and more about regular expressions, I kept widening the scope of my language to include more features. In the end, I decided on a much smaller set of features and finalized the design to accommodate those features. Although I had to settle for less than what I had set out to do, designing and implementing this little language has been a great learning experience in terms of understanding compiler and language design. The effect of my learning was especially felt while reading Stroustrup's design remarks he makes in his C++ language book about why he designed certain things in C++ the way he did.

7.2 Design a consistent error reporting mechanism

Many hours were wasted tracking down small, insignificant bugs because proper logging was not in place. I learned that a good error reporting module would have helped tremendously during the development cycle. Inserting print statements throughout the program made it very difficult in the end to replace them with newly designed logging facility which could have been avoided had I thought this through more carefully. If I had more time, I would have designed a logging and customer I/O library that will handle all kinds of message manipulation and control input from and output to the console as well as to a log file to facilitate debugging during the test cycle.

7.3 Write/design test cases as early as possible

I used many small examples for unit testing, but did not keep them around for regression testing purposes. This caused old bugs to be re-introduced and resulted in more wasted time fixing something that I had already fixed, but did not remember how I fixed it. If I had an automated test infrastructure in place early on, it would have been a breeze adding small unit tests to the test bucket and it would have made test development much easier later on. The quality of software is directly correlated to how well it was tested. Simply focusing too much on language design and not much on everything else cost me a bundle in the end.

7.4 Agile development

Adding to the first lesson I learned, I realized that my efforts to design everything up front did not yield much. In retrospect, an agile development style would have served me better especially in this time constrained development environment. By agile development, I mean design small, one or two features, develop and test and repeat this process by adding one new feature at a time.

8. Complete Listing

The complete listing of all files developed for this project is as follws:

```
total 120
-rw-r--r--   1 austin  austin  2817 Aug 11 13:57 Makefile
-rwxr-xr-x@  1 austin  austin  7020 Aug 11 13:32 TMLTreeWalkerHelper.cpp
-rwxr-xr-x@  1 austin  austin  1099 Aug 11 13:50 TMLTreeWalkerHelper.h
-rwxr-xr-x@  1 austin  austin  7655 Aug 11 16:06 grammar.g
-rw-r--r--@  1 austin  austin  2983 Aug 11 13:52 pattern.cpp
-rw-r--r--@  1 austin  austin  3686 Aug 11 13:51 pattern.h
-rw-r--r--@  1 austin  austin  3854 Aug 11 13:53 search.cpp
-rw-r--r--@  1 austin  austin  1888 Aug 11 13:51 search.h
drwxr-xr-x  14 austin  austin   476 Aug 11 22:01 test
-rw-r--r--@  1 austin  austin  2297 Aug 11 12:00 tml.cpp
-rwxr-xr-x@  1 austin  austin  1654 Aug 11 13:47 tml_common_defs.cpp
-rwxr-xr-x@  1 austin  austin  3180 Aug 11 13:47 tml_common_defs.h
-rw-r--r--@  1 austin  austin   475 Aug 11 12:58 tml_logger.cpp
-rw-r--r--@  1 austin  austin   599 Aug 11 11:56 tml_logger.h

./test:
total 8
-rwxr-xr-x   1 austin  austin  991 Aug 11 02:14 runtest.sh
drwxr-xr-x  14 austin  austin  476 Aug 11 22:01 tc0
drwxr-xr-x   5 austin  austin  170 Aug 11 22:01 tc1
drwxr-xr-x   5 austin  austin  170 Aug 11 22:01 tc10
drwxr-xr-x   5 austin  austin  170 Aug 11 22:01 tc2
drwxr-xr-x   5 austin  austin  170 Aug 11 22:01 tc3
drwxr-xr-x   5 austin  austin  170 Aug 11 22:01 tc4
drwxr-xr-x   5 austin  austin  170 Aug 11 22:01 tc5
drwxr-xr-x   5 austin  austin  170 Aug 11 22:01 tc6
drwxr-xr-x   5 austin  austin  170 Aug 11 22:01 tc7
drwxr-xr-x   5 austin  austin  170 Aug 11 22:01 tc8
drwxr-xr-x   5 austin  austin  170 Aug 11 22:01 tc9

./test/tc0:
total 96
-rw-r--r--  1 austin  austin  108 Aug 10 21:51 f1.txt
-rw-r--r--  1 austin  austin  522 Aug 11 02:09 f1.txt.out
-rw-r--r--  1 austin  austin  108 Aug 11 00:32 f2.txt
-rw-r--r--  1 austin  austin  522 Aug 11 02:09 f2.txt.out
-rw-r--r--  1 austin  austin  108 Aug 11 00:32 f3.txt
-rw-r--r--  1 austin  austin  522 Aug 11 02:09 f3.txt.out
-rw-r--r--  1 austin  austin  108 Aug 11 00:32 f4.txt
-rw-r--r--  1 austin  austin  522 Aug 11 02:09 f4.txt.out
-rw-r--r--  1 austin  austin  108 Aug 11 00:32 f5.txt
```

```
-rw-r--r--  1 austin  austin  522 Aug 11 02:09 f5.txt.out
-rw-r--r--  1 austin  austin   55 Aug 11 00:54 input.txt
-rw-r--r--  1 austin  austin  154 Aug 11 01:37 re0.tml

./test/tc1:
total 24
-rw-r--r--  1 austin  austin  140 Aug 10 22:20 input.txt
-rw-r--r--  1 austin  austin  111 Aug 11 02:09 input.txt.out
-rw-r--r--  1 austin  austin  146 Aug 10 22:18 re1.tml

./test/tc10:
total 24
-rw-r--r--  1 austin  austin  150 Aug 10 23:02 input.txt
-rw-r--r--  1 austin  austin  162 Aug 11 02:09 input.txt.out
-rw-r--r--  1 austin  austin  292 Aug 10 23:08 re10.tml

./test/tc2:
total 24
-rw-r--r--  1 austin  austin  140 Aug 10 22:30 input.txt
-rw-r--r--  1 austin  austin  141 Aug 11 02:09 input.txt.out
-rw-r--r--  1 austin  austin  192 Aug 10 22:29 re2.tml

./test/tc3:
total 24
-rw-r--r--  1 austin  austin  150 Aug 10 22:44 input.txt
-rw-r--r--  1 austin  austin   95 Aug 11 02:09 input.txt.out
-rw-r--r--  1 austin  austin  223 Aug 10 22:42 re3.tml

./test/tc4:
total 24
-rw-r--r--  1 austin  austin  150 Aug 10 22:45 input.txt
-rw-r--r--  1 austin  austin   95 Aug 11 02:09 input.txt.out
-rw-r--r--  1 austin  austin  269 Aug 10 22:47 re4.tml

./test/tc5:
total 24
-rw-r--r--  1 austin  austin  150 Aug 10 22:49 input.txt
-rw-r--r--  1 austin  austin  131 Aug 11 02:09 input.txt.out
-rw-r--r--  1 austin  austin  280 Aug 10 22:51 re5.tml

./test/tc6:
total 24
-rw-r--r--  1 austin  austin  140 Aug 10 22:52 input.txt
-rw-r--r--  1 austin  austin  104 Aug 11 02:09 input.txt.out
-rw-r--r--  1 austin  austin  183 Aug 10 22:54 re6.tml
```

./test/tc7:
total 24
-rw-r--r-- 1 austin austin 140 Aug 10 22:56 input.txt
-rw-r--r-- 1 austin austin 130 Aug 11 02:09 input.txt.out
-rw-r--r-- 1 austin austin 191 Aug 10 22:57 re7.tml

./test/tc8:
total 24
-rw-r--r-- 1 austin austin 150 Aug 10 22:58 input.txt
-rw-r--r-- 1 austin austin   44 Aug 11 02:09 input.txt.out
-rw-r--r-- 1 austin austin 246 Aug 10 22:59 re8.tml

./test/tc9:
total 24
-rw-r--r-- 1 austin austin 150 Aug 10 23:00 input.txt
-rw-r--r-- 1 austin austin   86 Aug 11 02:09 input.txt.out
-rw-r--r-- 1 austin austin 244 Aug 10 23:01 re9.tml

8.1 Antlr grammar - grammar.g

```
//
//Created by Austin Lee for COMS4115
//
header "pre_include_hpp"
{
}
header "post_include_hpp"
{
}
header "pre_include_cpp"
{
}
header
{
 #include "TMLTreeWalkerHelper.h"
 #include "tml_common_defs.h"
}
options
{
 language = "Cpp";
}
{
}

class TMLLexer extends Lexer;

options { testLiterals=false; k=2; }

ASSIGN          : '=' ;
//PARENS options { testLiterals = true; }
//          : '(' | ')' ;
LPAREN          : '(' ;
```

```
RPAREN          : ')' ;
LBRACE          : '{' ;
RBRACE          : '}' ;
LBRACK          : '[' ;
RBRACK          : ']' ;
ALTER  : '|' ;
COMMA           : ',' ;
SEMI    : ';' ;
PERIOD          : '.' ;
SQUOTE          : "'" ;
DASH   : '-' ;
DOLLAR          : '$' ;
AND             : '&' '&' ;
COMMENT : "//" (~('\r'|'\n'))* ('\r')? '\n' { _ttype = ANTLR_USE_NAMESPACE(antlr)Token::SKIP; } ;
protected LETTER
        : ('a' .. 'z' | 'A' .. 'Z') ;
protected DIGIT
        : '0' .. '9' ;

ID options { testLiterals = true; }
        : (LETTER | '_') (LETTER | DIGIT | '_')* ;
NUMBER          : (DIGIT)+ ;
STRING          : "'"! ("'" "'"! | ~("'"))* "'"! ; // '"
WS      : (' ' | '\t' | '\n' { newline(); } | '\r' )+ { _ttype = ANTLR_USE_NAMESPACE(antlr)Token::SKIP; }
;
ESC : '\\' (~('a' .. 'z'|'A' .. 'Z'|'0'..'9'))
        ;

class TMLParser extends Parser;
options { buildAST = true; k = 2; }

program : (statement SEMI!)* EOF!
        ;

statement
        : varDeclaration
        | copyStatement
        | regexpStatement
        ;

varDeclaration
        : varType ID^ (COMMA! ID)*
        ;

varType
        : "pattern" | "search" | "file" | "file_list" | "string"
        ;

patternExpression
        : (subPatternExprs)+
          {#patternExpression = #([PATTERNLIST,"PATTERNLIST"], patternExpression); }
        ;

subPatternExprs
        : LPAREN! (ID | compositePattern) groupPatternExprs RPAREN!
        ;
```

```
groupPatternExprs
        : (ALTER! (ID | compositePattern))*
          {#groupPatternExprs = #([GROUPLIST,"GROUPLIST"], groupPatternExprs); }
        ;

compositePattern
        : (LPAREN)? singletonPattern (patternQuantifier)? (RPAREN)?
        ;

singletonPattern
        : (stringExpression
        | classExpression)+
          {#singletonPattern = #([ATOMLIST,"ATOMLIST"], singletonPattern);}
        ;

stringExpression
        : characterExpression | STRING
        ;

//patternLocationSpecifier
//        : "beginsWith" | "bw"
//        | "endsWith" | "ew"
//        ;

patternQuantifier
        : "optional"
        | (NUMBER)? DASH (NUMBER)?
        ;

characterExpression
        : "dot"
        | "tab"
        | "bol"
        | "eol"
        | "space"
        | "ws"
        | "digit"
        | "nondigit"
        | "word"
        | "nonword"
        ;

classExpression
        : LBRACK! ("except")? charClassList RBRACK!
        ;

charClassList
        : (charClass)+
          {#charClassList = #([CHARCLASSLIST,"CHARCLASSLIST"], charClassList);}
        ;

charClass
        : ESC | NUMBER | ID | DOLLAR | DASH;

patternExpressionReference
```

```
        : "match"^ LBRACK! NUMBER RBRACK!
        | "lastmatch"^ LBRACK! NUMBER RBRACK!
        ;

//searchOptionDefinition
//      : "s_type" ASSIGN^ ("single" | "array")
//      | "s_input" ASSIGN^ ID
//      | "s_action" ASSIGN^ ("replace" | "delete")
//      | "s_mode" ASSIGN^ searchMode
//      | "s_pattern" ASSIGN^ ID
//      | "s_code" ASSIGN^ LBRACE! (ascii)* RBRACE!
//      | "s_replace" ASSIGN^ (ID | STRING)
//      ;

//searchMode
//      : "case"
//      | "nocase"
//      | "beginsWith"
//      | "endsWith"
//      | "reset"
//      ;

// <stuff on the left> = <stuff on the right>
copyStatement
        : ID^ (PERIOD^ modes)? ASSIGN^
        ( (stringLiteral) => thisStringLiteral
        | patternExpressionReference
        | patternExpression
        | ID )
        ;

stringLiteral
        : STRING
        ;

thisStringLiteral
        : STRING
        ;

modes
        : "s_mode"
        | "s_type"
        | "s_action"
        | "s_pattern"
        | "s_input"
        | "s_replace"
        | "noncapture"
        | "greedy"
        ;

regexpStatement
        : "regexp_engine"^ LPAREN! (ID)? RPAREN!
        ;

{
        #include <string>
```

```
        #include <iostream>
}

class TMLTreeWalker extends TreeParser;
{

}
program returns [int i]
{ int foo = 0; }
        : (foo=statement { i=foo; })*
        ;

statement returns [int i]
{
        i = 0;
        std::string vName, rval;
        char* vType;
        char* optionValue = NULL;
        Pattern rPattern,tmp;
}
          //case for copyStatement
        : #(ASSIGN
                (#(PERIOD left2:ID optionValue=flags { vName = left2->getText(); }) |
                                left1:ID { vName = left1->getText(); })
           (right2:STRING { rval = right2->getText(); setDefinition(vName,rval,optionValue); } |
                 (right1:ID { rval = right1->getText(); }
                 { setDefinition(vName,rval,optionValue); }) |
                 #(PATTERNLIST {rPattern.setVarName(vName);}
                   (tmp=subExpr { rPattern.addToSubPatternList(tmp);})+
                        { patternDefinition(rPattern); })
                ))

        | #(left:ID { vName = left->getText(); } vType=varType { declaration(vType,vName); }
           (right:ID { vName = right->getText(); declaration(vType,vName); })*)

        | #("regexp_engine" sName:ID { executeSearch(sName->getText()); } )

        ;

subExpr returns [Pattern p4]
{
        Pattern tmp2,tmp3;
        bool isGroup = false;
}
        : ((id1:ID
                {
                        Pattern tmp;
                        tmp.setVarName(id1->getText());
                        //cout << "id = " << id1->getText() << endl;
                        verifyPatternID(id1->getText());
                        resolveID(&tmp);
                        p4.addToSubPatternList(tmp);
                }
          | tmp2=patternExpr
                {
                        Pattern tmp4;
```

```
                        tmp2.constructPattern();
                        tmp4 = tmp2;
                        p4.addToSubPatternList(tmp4);
                }
                )
                (tmp3=groupExpr
                {
                        for( int i=0 ; i < tmp3.getPatternList().size() ; ++i )
                        {
                                p4.addToSubPatternList(tmp3.getPatternList().at(i));
                        }
                        p4.constructGroupPattern();
                        isGroup = true;
                })
                )
                {
                        if( !isGroup )
                        {
                                //p4.makeSubPattern();
                        }
                }
        ;

groupExpr returns [Pattern p3]
{
        Pattern tmp;
}
        : #(GROUPLIST
          ((id2:ID
                {
                        Pattern tmp1;
                        tmp1.setVarName(id2->getText());
                        //cout << "In group, id = " << id2->getText() << endl;
                        verifyPatternID(id2->getText());
                        resolveID(&tmp1);
                        p3.addToSubPatternList(tmp1);
                })
           | (tmp=patternExpr
                {
                        Pattern tmp2;
                        tmp.constructPattern();
                        tmp2 = tmp;
                        p3.addToSubPatternList(tmp2);
                })
           )*
           )
        ;

//compositePattern
patternExpr returns [Pattern p2]
{
        //Pattern ret,tmp;
        //ret.setLocationType(NONE);
        int a,b;
}
        : p2=subPatternExpr
```

```
		(("optional"
		  {
				a=0,b=1;
			} |
		  (from:NUMBER)? DASH (to:NUMBER)?
		  {
				if( from == NULL )
				{
					p2.setGreedy(false);
					a = -1;
				}
				else
					a = atoi(from->getText().c_str());

				if( to == NULL )
				{
					p2.setGreedy(true);
					b = -1;
				}
				else
					b = atoi(to->getText().c_str());
			})
		  { p2.setQuantifierOption(a,b);} )?
		  { }
		;

subPatternExpr returns [Pattern p1]
{
		char* t;
}
		: #(ATOMLIST (t=exprAtom
			{
				string tmp(t);
				p1.addToStrExprList(tmp);
			})+)
		;

exprAtom returns [char* t]
{
		char* tmp;
		std::string ret = "";
}
		: (("except" {ret="[^";})?
		  #(CHARCLASSLIST
		  (tmp=charClass
		  {
		    string tmp2(tmp);
			if( ret.length() == 0 )
				ret = "[";
			ret += tmp2;
		  })+) { ret += "]"; t = (char *)ret.c_str(); })
		| (tmp=singleChar
		  {
		    string tmp2(tmp);
			t = (char *)tmp2.c_str();
		  }
```

```
            { })
            ;

charClass returns [char* cc]
{

}
            : id1:ESC { cc = (char *)id1->getText().c_str(); }
            | id2:NUMBER { cc = (char *)id2->getText().c_str(); }
            | id3:ID { cc = (char *)id3->getText().c_str(); }
            | DOLLAR { cc = "$"; }
            | DASH { cc = "-"; }
            ;

singleChar returns [char* sc]
{
            sc = NULL;
}
            : "dot"                 { sc = "."; }
            | "tab"                 { sc = "\\t"; }
//          | "bol"                 { sc = "bol"; }
            | "eol"                 { sc = "\\n"; }
            | "space"           { sc = "\\s"; }
            | "ws"                  { sc = "\\s*"; }
            | "digit"           { sc = "\\d"; }
            | "nondigit"        { sc = "\\D"; }
            | "word"            { sc = "\\w"; }
            | "nonword"             { sc = "\\W"; }
            | str:STRING        { sc = (char *)str->getText().c_str(); }
            ;

varType returns [char* t]
            : "pattern" { t = "pattern"; }
            | "search" { t = "search"; }
            | "file" { t = "file"; }
            | "file_list" { t = "file_list"; }
            ;

flags returns [char* t]
            : "noncapture"      { t = "noncapture"; }
            | "greedy"              { t = "greedy"; }
            | "s_mode"              { t = "mode"; }
            | "s_input"             { t = "input"; }
            | "s_type"              { t = "type"; }
            | "s_pattern"       { t = "pattern"; }
            | "s_replace"       { t = "replace"; }
            | "s_action"        { t = "action"; }
            ;
```

8.2 Driver - tml.cpp

```
/*
 * Created by Austin Lee for COMS4115 PST
 *
 * Driver source that contains main for the TML program
```

```
 * Simply reads in the input file and invokes antlr
 * to tokenize, parse and generate the AST tree
 *
 */

#include <iostream>
#include <sstream>
#include <stdio.h>
#include <stdlib.h>
#include <string>
#include "antlr/AST.hpp"
#include "TMLLexer.hpp"
#include "TMLParser.hpp"
#include "TMLTreeWalker.hpp"
#include "CharInputBuffer.hpp"
#include "tml_common_defs.h"
#include "TMLTreeWalkerHelper.h"
#include "tml_logger.h"

//Singleton logger object to be used through the program
Logger* lg;

/*
 * 1st argument must be the .tml file to be translated/interpreted
 */
int main(int argc, char* argv[])
//int main()
{
	ANTLR_USING_NAMESPACE(std)
	ANTLR_USING_NAMESPACE(antlr)

	string filename = "re1.tml";
	FILE* inputFile = NULL;

	//0 - print only in debug
	//1 - always print
	lg = new Logger("tml_debug.txt",0);

	if( argc > 1 )
	{
		ostringstream ostr;
		istringstream tmpFileName;
		ostr << argv[1];
		tmpFileName.str(ostr.str());
		filename = tmpFileName.str();
	}
	else
	{
		cout << "Usage: tml <tml_file.tml>\n";
#ifndef DEBUG
		exit(0);
#endif
	}

	inputFile = fopen(filename.c_str(), "r");
	fseek(inputFile, 0L, SEEK_END);
```

```
	int size, readSize;
	size = ftell(inputFile);
	char* input = new char[size];
	fseek(inputFile, 0L, SEEK_SET);
	readSize = fread((void *)input, 1, size, inputFile);
	fclose(inputFile);
	if( size != readSize ) {
		cout << "Unable to read the whole input file\n";
		exit(0);
	}

	try
	{

		CharInputBuffer cib((unsigned char *)input,size);
		TMLLexer lexer(cib);
		lexer.setFilename(filename.c_str());
		TMLParser parser(lexer);
		parser.setFilename(filename.c_str());

		ASTFactory ast_factory;
		parser.initializeASTFactory(ast_factory);
		parser.setASTFactory(&ast_factory);

		parser.program();
		RefAST t = parser.getAST();
		if( t )
		{
			lg->printMsg(t->toStringTree(),0);
			TMLTreeWalker walker;
			int r = walker.program(t);
			lg->printMsg("TML execution result = ",r);
		}

		symbtable.cleanup();

	}
	catch(ANTLRException& e)
	{
		cerr << "Parse exception: " << e.toString() << endl;
		return -1;
	}
	catch(exception& e)
	{
		cerr << "exception: " << e.what() << endl;
		return -1;
	}

	delete [] input;
	delete lg;

	return 0;
}
```

8.3 TML common definitions – tml_common_defs.h, tml_common_defs.cpp

```
/*
 *  tml.h
 *  try
 *
 *  Created by Austin Lee on 7/18/08.
 *  Copyright 2008 __MyCompanyName__. All rights reserved.
 *
 */

#ifndef __tml_h
#define __tml_h

#include <stdio.h>
#include <stdlib.h>
#include <string>
#include <vector>
#include <map>
#include <iterator>

const int max_line_num = 256;

using namespace std;

/*
 * Class defintion for the symbol table
 *
 * I will use an associative map to maintain (key,element) pairs
 * key is the variable name and element is its type
 * access functions such as insert and lookup will be provide so that
 * if a different data structure other than the map is used in the future
 * the interface won't be affected
 */
class SymbolTable {

        map<string,string> _table;

public:

        //inserts identifier into the symbol table
        void insert(string key, string val);

        //checks to see if identifier id is already in the table
        bool lookup(string id);

        //clean up memory taken up by the symbol table
        void cleanup();

        string findValueByKey(string key);

};
```

```
typedef enum {
        TML_NO_ERROR=0,
        TML_FILE_NOT_FOUND,
        TML_SYNTAX_ERROR_INVALID_VALUE,
        TML_NO_MATCH_FOUND,
        TML_UNEXPECTED_ERROR
} TML_ERROR;

typedef struct {
        int from;
        int to;
} Quantifier;

typedef enum {
        CHAR = 0,
        DOT,
        TAB,
        BOL,
        EOL,
        SPACE,
        WS,
        DIGIT,
        NONDIGIT,
        WORD,
        NONWORD
} CharSet;

typedef struct {
        string strLiteral;
        CharSet ch;
        int ascii;
} StringLiteral;

typedef struct {
        int from;
        int to;
} CharacterClass;

typedef struct {
        int i;
        string cc;
        string s;
} PatternAtomType;

class MyBasicVar {
        string _name;

public:
        void setVarName(string varName) { _name = varName; }
        string getVarName() { return _name; }
};


class InputFile : public MyBasicVar {
```

```
	bool _isList;
	vector<string> _list;

public:

	//true = contains list of input files
	void setType(bool list=false) { _isList = list; }
	void setFileList(vector<string> list)
	{
		_list = list;
	}
	vector<string> getFileList()
	{
		//vector<string>::iterator pos;

		/*if( _list.size() == 0 )
		{
			printf("Error: No file defined\n");
			exit(0);
		}*/

		/*for( pos = _list.begin() ; pos != _list.end() ; ++pos )
		{
			list->push_back((*pos));
		}*/
		return _list;
	}

};

typedef enum {
	FIND_ONLY = 0,
	REPLACE,
//	DELETE
} SearchAction;


template <class T>
class ListHandler {

	vector<T*> _list;

public:

	//assumes that a check against the Symbol Table has been done
	//to determine this variable exists
	T* findByName(string p)
	{
		typename std::vector<T*>::iterator pos;
		T * ret = NULL;
		int i=0;
		for( pos = _list.begin() ; pos != _list.end() ; ++ pos, ++i )
		{
			if( p.compare( (*pos)->getVarName() ) == 0 )
				ret = _list.at(i);
		}
```

```
		return ret;
	}

	void remove(string p)
	{
		typename std::vector<T*>::iterator pos;
		T* ret = NULL;
		int i=0;
		for( pos = _list.begin() ; pos != _list.end() ; ++ pos, ++i )
		{
			if( p.compare( (*pos)->getVarName() ) == 0 )
			{
				ret = _list.at(i);
				_list.erase(pos);
			}
		}

		//this was allocated with new
		delete ret;
	}

	void add(T* p)
	{
		_list.push_back(p);
	}

	void cleanup()
	{

	}

};

string genOutFileName(string in);
char* getTMLError(int error);

#endif

/*
 *  tml_common_defs.cpp
 *  try
 *
 *  Created by Austin Lee on 7/31/08.
 *  Copyright 2008 __MyCompanyName__. All rights reserved.
 *
 */

#include <iostream>
#include <string>
#include <fstream>
#include <sstream>
#include "tml_common_defs.h"
#include "tml_logger.h"

extern Logger* lg;
```

```
using namespace std;
//using namespace boost;

/*
 * inserts identifier into the symbol table
 * key is the identifier and val is the typename
 */
void SymbolTable::insert(string key, string val)
{
	_table.insert(map<string,string>::value_type(key,val));
}

//checks to see if identifier id is already in the table
bool SymbolTable::lookup(string id)
{
	//map<string,string>::iterator pos;

	if( _table.find(id) == _table.end() )
		return false;
	else
		return true;
}

string SymbolTable::findValueByKey(string key)
{
	map<string,string>::iterator pos;
	for( pos = _table.begin() ; pos != _table.end() ; ++pos )
	{
		if( pos->first == key )
			return pos->second;
	}

	return "";
}

void SymbolTable::cleanup()
{
	//remove all elements
	//_table.clear();
	//free memory
	//_table.~map();
}

string genOutFileName(string in)
{
	string ret;

	ret = in + ".tmp";

	return ret;
}


char* getTMLError(int error)
{
```

```
	TML_ERROR e = (TML_ERROR)error;
	char* ret = NULL;
	switch (e)
	{
		case TML_NO_ERROR:
			break;
		case TML_FILE_NOT_FOUND:
			ret = "File Not Found";
			break;
		case TML_SYNTAX_ERROR_INVALID_VALUE:
			ret = "Syntax Error - Invalid value";
			break;
		case TML_NO_MATCH_FOUND:
			ret = "No Match Found for Given Pattern";
			break;
		case TML_UNEXPECTED_ERROR:
		default:
			ret = "Unknown Error";
			break;
	}//end of switch

	return ret;
}
```

8.4 Pattern class – pattern,h, pattern.cpp

```
/*
 *  pattern.h
 *  tml
 *
 *  Created by Austin Lee.
 *
 */

#ifndef __PATTERN_H
#define __PATTERN_H

#include "tml_common_defs.h"

class Pattern : public MyBasicVar {

	bool _capture;
	bool _greedy;
	bool _group;    //true = patterns are alternatives (p1 | p2 | p3), false is default
	bool _charClass;  //treat the pattern as a character class
	bool _begins;
	bool _ends;

	string _regexp;

	Quantifier _qf;
	CharacterClass _class;
	vector<Pattern> _groupList;
	//vector<Pattern> _andList;

	//a pattern can be a sequence of patterns
```

```
        vector<Pattern> _patternList;
        //sequence of string literals
        vector<StringLiteral> _sl;
        vector<string> _strExprList;

public:

        Pattern()
        {
                _capture=true;
                _greedy=true;
                _group=false;
                _charClass=false;
                _begins=false;
                _ends=false;
                _qf.from=-1;
                _qf.to=-1;
        }

        void setCapture(bool capture=true) { _capture = capture; }
        void setGreedy(bool greedy=true) { _greedy = greedy; }
        void setGroup(bool group=false) { _group = group; }
        void setClass(bool isClass=false) { _charClass = isClass; }

        void setStringLiteral(string str);
        void setStringLiteral(vector<StringLiteral> str)
        {
                _sl.clear();
                copy(str.begin(),str.end(),back_inserter(this->_sl));
        }

        void setPatternList(vector<Pattern> plist)
        {
                _patternList.clear();
                copy(plist.begin(),plist.end(),back_inserter(this->_patternList));
        }
        //void setLocationType(LocatioinType loc=NONE) { _loc = loc; }
        void setRegExp(string str) { _regexp = str; }

        string getRegExp() { return _regexp; }

        void setQuantifierOption(int from, int to) { _qf.from = from; _qf.to = to; }
        void setCharacterClass(int from, int to) { _class.from = from; _class.to = to; }

        vector<Pattern> getPatternList()
        {
                vector<Pattern> tmp;
                copy(_patternList.begin(),_patternList.end(),back_inserter(tmp));
                return tmp;
        }

        //checks to see if the pattern is valid or not
        //might want to return some debugging information so user knows what to fix..
        //bool isValidPattern(Pattern p);
        //bool isValidPattern();
```

```
void addToStrExprList(string str)
{
	_strExprList.push_back(str);
}

int addToSubPatternList(Pattern p)
{
	Pattern tmp;
	tmp = p;
	_patternList.push_back(tmp);
	return 0;
}
int addToGroupList(Pattern p)	{ _groupList.push_back(p); return 0; }
//int addToAndList(Pattern p)	{ _andList.push_back(p); return 0; }

void printPatternList();

string constructPattern();
string constructStringLiteral();
//string constructClassPattern();
string constructGroupPattern();
string concatPatternList();

void makeSubPattern()
{
	_regexp = "(" + _regexp + ")";
}

Pattern& operator= (const Pattern& p)
{
	_capture   = p._capture;
	_greedy    = p._greedy;
	_group     = p._group;
	_charClass = p._charClass;
	_regexp    = p._regexp;
	_qf        = p._qf;
	_class     = p._class;
	this->_regexp    = p._regexp;
	//_groupList = p._groupList;
	//_patternList = p._patternList;
	//_sl        = p._sl;
	this->setVarName(((Pattern)p).getVarName());
	this->_groupList.clear();
	this->_patternList.clear();
	this->_strExprList.clear();
	copy(p._groupList.begin(),p._groupList.end(),back_inserter(this->_groupList));
	copy(p._patternList.begin(),p._patternList.end(),back_inserter(this->_patternList));
	copy(p._strExprList.begin(),p._strExprList.end(),back_inserter(this->_strExprList));
	//copy(p._sl.begin(),p._sl.end(),back_inserter(this->_sl));
	vector<StringLiteral>::const_iterator pos = (p._sl).begin();
	for( ; pos != (p._sl).end() ; ++pos )
	{
		StringLiteral tmp;
		tmp.strLiteral = (*pos).strLiteral;
		tmp.ch = (*pos).ch;
		tmp.ascii = (*pos).ascii;
```

```
                        _sl.push_back(tmp);
                }
                return *this;
        }
};

#endif

/*
 *  pattern.cpp
 *  tml2
 *
 *  Created by Austin Lee on 8/11/08.
 *  Copyright 2008 __MyCompanyName__. All rights reserved.
 *
 */

#include <sstream>
#include "pattern.h"
#include "tml_logger.h"

extern Logger* lg;

void Pattern::setStringLiteral(string str)
{
        StringLiteral sl;
        sl.strLiteral = str;
        _sl.push_back(sl);
}

string Pattern::constructPattern()
{
        string ret;

        vector<string>::const_iterator pos;
        for( pos = _strExprList.begin() ; pos != _strExprList.end() ; ++pos )
        {
                if( *pos != "" )
                        ret += *pos;
                else
                {
                        //Error!
                        lg->printMsg("In constructPattern
",(int)TML_SYNTAX_ERROR_INVALID_VALUE);
                }
        }

        ostringstream ostr;

        if( _qf.from != -1 || _qf.to != -1 )
        {
                int a,b;
                a = _qf.from;
                b = _qf.to;
                if( a != -1 && b != -1 )
                        ostr << "{" << a << "," << b << "}";
```

```
                else if( a == -1 )
                        ostr << "{0," << b << "}";
                else if( b == -1 )
                        ostr << "{" << a << ",}";
                else
                {
                        //Error
                        lg->printMsg("In constructPattern
",(int)TML_SYNTAX_ERROR_INVALID_VALUE);
                }
                if( !_greedy )
                        ostr << "?";
        }

        ret += ostr.str();

        _regexp = ret;

        return ret;
}

string Pattern::constructGroupPattern()
{
        string ret = "(";
        string regexp;

        vector<Pattern>::iterator pos;

        for( pos = _patternList.begin() ; pos != _patternList.end() ; ++pos)
        {
                //strip "(" and ")" before adding to the group
                //to avoid creating an unnecessary subpattern
                //"(" and ")" will be applied to the whole group
                //so that the alternative that gets matched becomes
                //the subpattern for the group
                if( (*pos)._regexp.substr(0,1).compare("(") == 0 )
                        regexp = (*pos)._regexp.substr(1,(*pos)._regexp.length() - 2);
                else
                        regexp = (*pos)._regexp;

                if( pos == _patternList.begin() )
                        ret += regexp;
                else
                        ret += "|" + regexp;
        }

        ret += ")";
        _regexp = ret;
        return ret;
}

string Pattern::concatPatternList()
{
        string ret;

        vector<Pattern>::iterator pos;
```

```
	for( pos = _patternList.begin() ; pos != _patternList.end() ; ++pos)
	{
		ret += (*pos)._regexp;
	}

	this->_regexp = ret;
	return ret;
}

void Pattern::printPatternList()
{
	vector<Pattern>::iterator pos;
	for( pos = _patternList.begin() ; pos != _patternList.end() ; ++pos)
	{
		if( (*pos)._patternList.size() > 0 )
		{
			(*pos).printPatternList();
		}
		else
		{
			string msg = "(" + (*pos).getVarName() + ": " + (*pos)._regexp + ")";
			lg->printMsg(msg,0);
		}
	}
}

string Pattern::constructStringLiteral()
{
	string ret;
	vector<StringLiteral>::const_iterator pos;
	for( pos = _sl.begin() ; pos != _sl.end() ; ++pos )
	{
		if( (*pos).strLiteral != "" )
			ret += (*pos).strLiteral;
	}

	ostringstream ostr;

	if( _qf.from != -1 || _qf.to != -1 )
	{
		int a,b;
		a = _qf.from;
		b = _qf.to;
		if( a != -1 && b != -1 )
			ostr << "{" << a << "," << b << "}";
		else if( a == -1 )
			ostr << "{0," << b << "}";
		else if( b == -1 )
			ostr << "{" << a << ",}";
		else
		{
			//Error
		}
		if( !_greedy )
			ostr << "?";
```

```
		}

		ret += ostr.str();

		return ret;
}
```

8.5 Search class – search.h, search.cpp

```
/*
 *  search.h
 *  tml
 *
 *  Created by Austin Lee.
 *
 */

#ifndef __SEARCH_H
#define __SEARCH_H

#include "tml_common_defs.h"
#include "pattern.h"

class Search : public MyBasicVar {
		bool _case;		//true = case-sensitivte (default), false = case-insensitive
		bool _singleLine;	//true = treat input line as single line, false = treat input as multi-line
		bool _reset;		//true = set the /G option, false is default
		bool _bw;
		bool _ew;

		//string _code;
		Pattern* _p;
		vector<string> _matched;
		SearchAction _action;
		string _replace;
		InputFile* _input;

		string constructPattern();

public:

		void setDefaults()
		{
			setSearchAction(FIND_ONLY);
			setCase(true);
			setLineMode(true);
			setResetMode(false);
			setBeginsWith(false);
			setEndsWith(false);
		}

		void setCase(bool caseSensitive=true) { _case = caseSensitive; }
		void setLineMode(bool mode=true) { _singleLine = mode; }
		void setResetMode(bool mode=false) { _reset = mode; }
		void setPattern(Pattern *p) { _p = p; }
		void setSearchAction(SearchAction a=FIND_ONLY) { _action = a; }
		void setReplaceString(string str) { _replace = str; }
		void setInput(InputFile* in) { _input = in; }
		void setBeginsWith(bool b=false) { _bw=b; }
		void setEndsWith(bool e=false) { _ew=e; }
```

```
		//void setCode(string code) { _code = code; }
		//void executeCodeSection();

		//Input can be a string or a file or a list of files
		int performSearch();
		//int performSearch(Input *in);
		//int performSearch(Input *in, Output *out);  //might be necessary..

		//the following two use _matched for backreferencing
		//string getMatched(int index);
		//string getLastMatched(int index);

		Pattern* getPattern() { return _p; }
		string getReplaceString()
		{
			//return a copy, not the actually thing
			string tmp(_replace);
			return tmp;
		}

		InputFile* getInput() { return _input; }
		bool isReset() { return _reset; }

};

#endif

/*
 *  search.cpp
 *  tml
 *
 *  Created by Austin Lee.
 *
 */

#include <fstream>
#include <sstream>
#include "search.h"
#include "boost/regex.hpp"
#include "tml_logger.h"

extern Logger* lg;

using namespace std;

//performs search on this
int Search::performSearch()
{
		int ret=0;
		bool found=false;

		//use Perl style regexp
		boost::regex::flag_type mType = boost::regex::perl;
		//if case-insensitive
		if( _case == false )
			mType = mType | boost::regex::icase;

		string p = getPattern()->getRegExp();

		//for testing only
		//p = "(word\\s)";
```

```
//string r = " found ";
//setReplaceString(r);

if( _bw )
        p = "^" + p;
if( _ew )
        p = p + "$";

lg->printMsg("Regular Expression:",0);
lg->printMsg(p,0);

boost::regex reg(p,mType);

vector<string> filelist;
if( getInput() != NULL )
        filelist = getInput()->getFileList();
else
{
        lg->printMsg("Error: no input file was specified for Search ",0);
        lg->printMsg(getVarName(),0);
        exit(0);
}

vector<string>::iterator pos;
std::ifstream from;
std::ofstream to;
char str[max_line_num];
memset(str,0,max_line_num);
int line_num,position;

string msg;
ostringstream ostr;

for( pos = filelist.begin() ; pos != filelist.end() ; ++pos )
{
        from.open((*pos).c_str(),ios_base::in);
        if( from.is_open() == false )
                lg->printMsg("In performSearch",(int)TML_FILE_NOT_FOUND);

        string outFilename = genOutFileName((*pos).c_str());
        to.open(outFilename.c_str(),ios_base::out | ios_base::trunc);
        if( to.fail() )
        {
                lg->printMsg("In performSearch",(int)TML_FILE_NOT_FOUND);
                exit(0);
        }

        line_num = 1;

        while( from.getline(str,max_line_num) )
        {
                string s(str);
                switch( _action )
                {
                        case FIND_ONLY:
                        {
                                boost::smatch m;
                                boost::match_flag_type flags = boost::match_default;

                                //if isReset = true, then we search the rest of the input for more
matches
                                if( isReset() )
```

```
                                        {
                                                string::const_iterator pos = s.begin();
                                                string::const_iterator e_pos = s.end();
                                                int repeat=1;
                                                int subMatchCount = 0;


                                                while( boost::regex_search(pos,e_pos,m,reg,flags) )
                                                {
                                                        subMatchCount = m.size();

                                                        if( repeat )
                                                        {
                                                                ostr << "Line " << line_num << ":";
                                                                repeat = 0;
                                                        }

                                                        for( int i=1 ; i <= subMatchCount ; ++i )
                                                        {
                                                                //this is only if sub-patterns are
matched

                                                                if( m[i].matched )
                                                                {
                                                                        string
match_str(m[i].first,m[i].second);
                                                                        position = (m[i].first -
s.begin())+1;

                                                                        msg = p + " matched " +
m[i].str();

                                                                        lg->printMsg(msg,0);

                                                                        ostr << " (pos: " << position
<< ") " << m[i];
                                                                }
                                                        }
                                                        //go to the end of the current match
                                                        pos=m[0].second;
                                                }
                                        }
                                        else
                                        {
                                                found = boost::regex_search(s, m, reg, flags);
                                                //here we only care about the first match
                                                if( m[0].matched )
                                                {
                                                        string match_str(m[0].first,m[0].second);
                                                        position = (m[0].first - s.begin())+1;

                                                        msg = p + " matched " + m[0].str();
                                                        lg->printMsg(msg,0);

                                                        ostr << "Line " << line_num  << ": (pos: " <<
position << ") " << m[0];
                                                }
                                        }
                                        break;
                                }
                                case REPLACE:
                                {
                                        string replaceStr = getReplaceString();
                                        if( replaceStr.length() == 0 )
```

```
                                        {
                                                //Error - no replace string provided
                                        }
                                        s = boost::regex_replace(s, reg, replaceStr);
                                        ostr << s;
                                        break;
                                }
                                        //case DELETE:
                                        //      break;
                                default:
                                        break;
                        }

                        //if a match was found
                        if( ostr.str().length() > 0 )
                        {
                                ostr << endl;
                                s = ostr.str();
                                to.write(s.c_str(),s.size());
                                to.flush();
                        }
                        memset(str,     0, max_line_num);
                        ++line_num;
                        //reset the outstream string buffer
                        ostr.str("");
                }

                from.close();
                to.close();
        }

        return ret;
}
```

8.6 Tree traverse & helper functions – TMLTreeWalkerHelper.h, TMLTreeWalkerHelper.cpp

```
/*
 *  TMLTreeWalkerHelper.h
 *  try
 *
 *  Created by Austin Lee on 7/31/08.
 *  Copyright 2008 __MyCompanyName__. All rights reserved.
 *
 */

#ifndef __TMLTreeWalkerHelper_h
#define __TMLTreeWalkerHelper_h

#include "tml_common_defs.h"
#include "search.h"
#include "pattern.h"

//only need one for the language
static SymbolTable symbtable;

/*****************************************************************************
 *              Functions needed by TMLWalker.cpp
```

```
 ********************************************************************************/

void declaration(char* varTypeName, string id);

void setDefinition(string varName, string rVal, char* optionValue);

void fileDefinition(string id, string rvalue, bool isList);

void patternDefinition(string lval, string rval,char* optionValue);
void patternDefinition(Pattern p);

void searchDefinition(string lval, string rval,char* optionValue);
void searchOptionDefinition(string lval, string option, string rval);

void verifyPatternID(string pName);
void resolveID(Pattern* p);

void executeSearch(string s);

#endif

/*
 *  TMLTreeWalkerHelper.cpp
 *  try
 *
 *  Created by Austin Lee on 7/31/08.
 *  Copyright 2008 __MyCompanyName__. All rights reserved.
 *
 */

#include <iostream>
#include <sstream>
#include <fstream>
#include <string>
#include "TMLTreeWalkerHelper.h"
#include "tml_logger.h"

extern Logger* lg;

using namespace std;

ListHandler<Pattern> patternListHandler;
ListHandler<Search>  searchListHandler;
ListHandler<InputFile> inputFileListHandler;

void declaration(char* varTypeName, string id)
{
	string msg(varTypeName);
	msg += " " + id;
	lg->printMsg(msg,0);

	if( true == symbtable.lookup(id))
	{
		lg->printMsg(id, (int)TML_SYNTAX_ERROR_INVALID_VALUE);
	}
	else {
```

```
                string s(varTypeName);
                symbtable.insert(id, s);
        }
}

void setDefinition(string varName, string rval, char* optionValue)
{
        if( true == symbtable.lookup(varName) )
        {
                string typeName;
                typeName = symbtable.findValueByKey(varName);

                if( typeName.compare("pattern") == 0 )
                {
                        patternDefinition(varName, rval,optionValue);
                }
                else if( typeName.compare("search") == 0 )
                {
                        searchDefinition(varName, rval,optionValue);
                }
                else if( typeName.compare("file") == 0 )
                {
                        fileDefinition(varName, rval, false);
                }
                else if( typeName.compare("file_list") == 0 )
                {
                        fileDefinition(varName, rval, true);
                }
                else
                {
                        lg->printMsg("In setDefinition",(int)TML_UNEXPECTED_ERROR);
                }

        }
        else
        {
                cout << "Error: " << varName << " not found\n";
                string msg(varName);
                msg = "In setDefinition " + msg;
                lg->printMsg(msg,(int)TML_SYNTAX_ERROR_INVALID_VALUE);
        }
}

void fileDefinition(string lval, string rvalue, bool isList)
{
        bool found=false;
        InputFile* in;
        if( (in = inputFileListHandler.findByName(lval)) == NULL )
        {
                in = new InputFile;
                in->setType(false);
                in->setVarName(lval);
                found = false;
        }
        else
                found = true;
```

```
		vector<string> list;
		if( !isList )
		{
			list.push_back(rvalue);
		}
		else
		{
			ifstream from;
			from.open(rvalue.c_str(),ios_base::in);
			char filename[max_line_num];
			while( from.getline(filename,max_line_num) )
			{
				string s(filename);
				list.push_back(s);
			}

		}

		in->setFileList(list);

		if( !found )
		{
			inputFileListHandler.add(in);
		}

}

/*
 * Pattern is defined as a simple string literal
 */
void patternDefinition(string lval, string rval,char* optionValue)
{
		Pattern* p = new Pattern;
		string regexp;
		p->setVarName(lval);
		p->setStringLiteral(rval);
		regexp = "(" + rval + ")"; //make the string literal a subpattern
		p->setRegExp(regexp);
		patternListHandler.add(p);
}

void patternDefinition(Pattern myP)
{
		Pattern p;
		p = myP;

		//traverse the subpattern list and resolve all references to already defined patterns
		vector<Pattern>::iterator itr1,itr2;
		vector<Pattern> patternList = p.getPatternList();
		for( itr1 = patternList.begin() ; itr1 != patternList.end() ; ++itr1 )
		{
			vector<Pattern> patternList2 = (*itr1).getPatternList();
			for( itr2 = patternList2.begin() ; itr2 != patternList2.end() ; ++itr2 )
			{
				string patternRef = (*itr2).getVarName();
```

```
                        //if name is set, then it must be a reference
                        if( patternRef.size() > 0 )
                        {
                                if( true == symbtable.lookup(patternRef) )
                                {
                                        //if found, make a copy of it
                                        Pattern* tmp = patternListHandler.findByName(patternRef);
                                        (*itr2) = (*tmp);
                                }
                                else
                                {
                                        lg->printMsg("In
patternDefinition",(int)TML_SYNTAX_ERROR_INVALID_VALUE);
                                }
                        }
                }

                (*itr1).setPatternList(patternList2);
        }

        p.setPatternList(patternList);
        p.printPatternList();

        string msg = p.concatPatternList();
        lg->printMsg("Concatenated pattern = ",0);

        Pattern* pNew = new Pattern();
        (*pNew) = p;

        string pName = p.getVarName();
        //if this pattern has already been defined, redefine it
        if( patternListHandler.findByName(pName) != NULL )
                patternListHandler.remove(pName);
        patternListHandler.add(pNew);
}

/*
 * when setting one search to another
 * Just make another copy of it
 */
void searchDefinition(string lval, string rval,char* optionValue)
{
        if( true == symbtable.lookup(lval) )
        {
                if( symbtable.findValueByKey(lval).compare("search") == 0 )
                {
                        Search* sch;
                        bool isNew=false;
                        //check search_list to make sure an object for this does not already exists
                        if( (sch = searchListHandler.findByName(lval)) == NULL )
                        {
                                isNew = true;
                                sch = new Search;
                                sch->setVarName(lval);
                                sch->setDefaults();
                        }
```

```
                        //check to see if rval is a reference
                        //if so, check to see if type matches

                        //otherwise, set option flag

                        string s(optionValue);
                        if( s.compare("input") == 0 )
                        {
                                InputFile* in = inputFileListHandler.findByName(rval);
                                sch->setInput(in);
                        }
                        else if( s.compare("pattern") == 0 )
                        {
                                if( symbtable.findValueByKey(rval).compare("pattern") == 0 )
                                {
                                        Pattern* p;
                                        p = patternListHandler.findByName(rval);
                                        sch->setPattern(p);
                                }
                                else
                                {
                                        //Error: this pattern variable is not in Symbol Table
                                        lg->printMsg("In searchDefinition
",(int)TML_SYNTAX_ERROR_INVALID_VALUE);
                                }
                        }
                        else if( s.compare("action") == 0 )
                        {
                                if( rval.compare("replace") == 0 )
                                        sch->setSearchAction(REPLACE);
                                else
                                {
                                        //Error
                                        lg->printMsg("In searchDefinition
",(int)TML_SYNTAX_ERROR_INVALID_VALUE);
                                }
                        }
                        else if( s.compare("replace") == 0 )
                        {
                                sch->setReplaceString(rval);
                        }
                        //else if( s.compare("type") == 0 )
                        //{
                        //}
                        else if( s.compare("mode") == 0 )
                        {
                                if( rval.compare("case") == 0 )
                                {
                                        sch->setCase(true);
                                }
                                else if( rval.compare("nocase") == 0 )
                                {
                                        sch->setCase(false);
                                }
                                else if( rval.compare("beginsWith") == 0 )
                                {
```

```
                                        sch->setBeginsWith(true);
                                }
                                else if( rval.compare("endsWith") == 0 )
                                {
                                        sch->setEndsWith(true);
                                }
                                else if( rval.compare("reset") == 0 )
                                {
                                        sch->setResetMode(true);
                                }
                                else
                                {
                                        //error, invalid rvalue
                                        lg->printMsg("In searchDefinition
",(int)TML_SYNTAX_ERROR_INVALID_VALUE);
                                }

                        }
                        else //Error!
                        {
                                lg->printMsg("In searchDefinition
",(int)TML_SYNTAX_ERROR_INVALID_VALUE);
                        }

                        if( isNew )
                                searchListHandler.add(sch);

                }
                else
                {
                        //wrong type, error
                        lg->printMsg("In searchDefinition
",(int)TML_SYNTAX_ERROR_INVALID_VALUE);
                }
        }
        else
        {
                //error
                lg->printMsg("In searchDefinition
",(int)TML_SYNTAX_ERROR_INVALID_VALUE);
        }

}

/*
 * Verified that the pattern id being used on the right side
 * of a pattern definition has already been defined
 */
void verifyPatternID(string pName)
{
        if( patternListHandler.findByName(pName) == NULL )
        {
                lg->printMsg("Error: undefined pattern
identifier",(int)TML_SYNTAX_ERROR_INVALID_VALUE);
                lg->printMsg(pName,0);
                exit(0);
```

```
		}
}

/*
 * Resolve the reference by making a copy
 */
void resolveID(Pattern* p)
{
		Pattern* tmp = patternListHandler.findByName(p->getVarName());
		(*p) = (*tmp);
}

void executeSearch(string s)
{
		Search* sch;
		int ret=0;
		sch = searchListHandler.findByName(s);
		ret = sch->performSearch();
}
```

8.7 Logger

```
/*
 *  tml_logger.h
 *  tml
 *
 *  Created by Austin Lee.
 *
 */

#ifndef __TML_LOGGER
#define __TML_LOGGER

#include <iostream>
#include <string>

class Logger {

		/*
		 * -1 = print nothing
		 * 0 = print only in debug
		 * 1 = print always
		 * 2 = write to a log file as well
		 */
		int _logLevel;
		char* _logfilename;
		std::string _msg;
		int _error;

public:

		Logger(std::string filename, int level)
		{
				_logfilename = (char *)filename.c_str();
				_logLevel = level;
```

```
        }
        void setLogLevel(int i);

        void printMsg(char* s, int error=0);
        void printMsg(std::string s, int error=0);

};

#endif

/*
 *  tml_logger.cpp
 *  tml
 *
 *  Created by Austin Lee.
 *
 */

#include <iostream>
#include "tml_logger.h"
#include "tml_common_defs.h"

void Logger::printMsg(char* msg, int error)
{

#ifdef DEBUG
        if( _logLevel >= 0 )
#else
        if( _logLevel >= 1 )
#endif
        {
                if( error != 0 )
                        std::cout << msg << " " << getTMLError(error) << endl;
                else
                        std::cout << msg << endl;
        }
}

void Logger::printMsg(std::string msg, int error)
{
        printMsg((char *)msg.c_str(), error);
}
```

8.8 Makefile

```
#################################################################################
#
# Created by Austin Lee
# for COMS4115, Columbia University
# E-mail: taehee@gmail.com
#
# Settings are specific to Mac OS X Leopard v10.5.4
# Assumptions:
# 1. Both Boost.Regex and antlr v2 are installed on the system
# 2. Both libraries are already in the system library path
```

```
# 3. Using GNU's gcc compiler (g++)
#
##############################################################################

BOOST_DIR = boost-1_35
ANTLR_DIR = antlr

# do not build with debug boost library by default
# both of these libraries get built and installed
# with standard boost install
ifdef DEBUG_BOOST
BOOST_LIB = boost_regex-gccD
else
BOOST_LIB = boost_regex-gcc
endif

# It is not recommended to use debug version of antlr
# But to link with debug antlr lib, the source will
# need be reconfigured with --enable-debug
# and rebuilt with debug=1
# antlr will not name debug version differently so
# the resulting lib will need to be renamed to antlrD
ifdef DEBUG_ANTLR
ANTLR_LIB = antlrD
else
ANTLR_LIB = antlr
endif

ifndef SYS_INC_DIR
  SYS_INC_DIR = /usr/local/include
endif

INC = -I. -I$(SYS_INC_DIR)/$(BOOST_DIR) -I$(SYS_INC_DIR)/$(ANTLR_DIR)
LIB = -l$(BOOST_LIB) -l$(ANTLR_LIB)
CXX := g++
CXX_OUT_FLAG := -o
CXX_FLAGS = -O2 -felide-constructors -pipe
LD := g++
LD_OUT_FLAG := -o
LD_FLAGS :=

ifdef DEBUG
  PRODUCT_MODE = debug
  CXX_FLAGS += -Wall -DDEBUG -D_DEBUG
else
  PRODUCT_MODE = release
  CXX_FLAGS += -DNDEBUG
endif

CXX_FLAGS += -c

ROOT = `pwd`/..

TML_DIR := $(ROOT)/tml/$(PRODUCT_MODE)/
OBJ_DIR := $(ROOT)/obj/$(PRODUCT_MODE)/
SRC_DIR := $(ROOT)/src/
```

```
MKDIRS = mkdirs

#ANTLR_JAR = $(ROOT)/antlr/antlr.jar
ANTLR_CMD = antlr

TMLGrammarFile := grammar.g
TMLSrcFiles = TMLLexer.cpp TMLParser.cpp TMLTreeWalker.cpp \
		pattern.cpp search.cpp tml_logger.cpp \
		tml_common_defs.cpp TMLTreeWalkerHelper.cpp tml.cpp
TMLObjFiles = TMLLexer.o TMLParser.o TMLTreeWalker.o \
		pattern.o search.o \
		tml_common_defs.o TMLTreeWalkerHelper.o tml.o tml_logger.o
TML := $(TML_DIR)tml
GRAM := gram

TMLSRCFILES = $(addprefix $(SRC_DIR),$(TMLSrcFiles))
TMLOBJFILES = $(addprefix $(OBJ_DIR),$(TMLObjFiles))

foo := $(shell echo $(TMLSrcFiles))

all: $(GRAM) $(MKDIRS) $(TML)

gram:
	$(ANTLR_CMD) $(TMLGrammarFile)

mkdirs:
	mkdir -pv $(TML_DIR)
	mkdir -pv $(OBJ_DIR)

.PHONY: test
test:
	rm -f -r $(ROOT)/test
	cp -f -r $(ROOT)/src/test ../
	cp $(TML) ../test

$(TML): $(MKDIRS) $(TMLObjFiles)
	$(LD) $(LD_FLAGS) $(LD_OUT_FLAG) $@ $(TMLOBJFILES) $(LIB)

$(TMLObjFiles): %.o: %.cpp
	$(CXX) $(CXX_FLAGS) $< $(CXX_OUT_FLAG) $(OBJ_DIR)$@ $(INC)

.PHONY: clean
clean:
	rm -f -r $(OBJ_DIR)*.o $(TML) TMLLexer.cpp TMLLexer.hpp \
		TMLParser.cpp TMLParser.hpp TMLTreeWalker.cpp \
		TMLTreeWalker.hpp TMLLexerTokenTypes.* *~
```

8.9 Test script

The .tml and .txt files are omitted here, but they will be part of the project submission.

runtest.sh

```
#!/bin/sh
# Created by Austin Lee for COMS4115

echo "Running TML regression test suites"

tml=`pwd`/tml
if [ -n "$1" ]
then
  count="$1"
else
  count=1
fi

tc="tc"
index=1
while [ $index -le $count ]
do
  echo "Running TC$index ..."
  $tml "$tc$index/re$index.tml"
  if [ -e "$tc$index/input.txt.tmp" ]
  then
    res=`diff $tc$index/input.txt.out $tc$index/input.txt.tmp`
    if [ -n "$res" ]
      then
        echo "Failed" > "result$index.txt"
      else
        echo "Passed" > "result$index.txt"
      fi
    else
      echo "Failed, no output generated" > "result$index.txt"
    fi
  index=$(($index+1))
done

index=1
filecount=5
echo "Running the default file_list TC..."
$tml "tc0/re0.tml"
while [ $index -le $filecount ]
do
  res=`diff tc0/f$index.txt.out tc0/f$index.txt.tmp`
```

```
  if [ -n "$res" ]
  then
    echo "$index failed" >> "result0.txt"
  else
    echo "$index passed" >> "result0.txt"
  fi
  index=$(($index+1))
done

echo "Tests completed"

echo "Test Results"
grep -i pass result*
grep -i fail result*
```