**The Lemon Parser C# Translator**

This program converts a parser file created by the Lemon Parser into C# code. When combined with two other C# files, can be compiled into an assembly or directly incorporated into a program.

The design goals were to:

1. Create a thread-safe C# parser without wrappers or unsafe code.
2. Omit C code generated by the Lemon Parser that duplicates built in .NET features such as stacks and memory management, or that can be changed to a simpler object oriented approach.
3. Keep it understandable when comparing a c and corresponding C# parsers. Variable names are unchanged between the versions. Method names are changed slightly. I could have translated variable names to a “modern style.” However, this obfuscated the process of translation.
4. Use classes to define the user's internal data representations. The Lemon Parser includes directives to accommodate special data representations. However, allowing the user to specify a class provides greater flexibility and separates the parser from the data representation.
5. Don't obsess over memory. The Lemon Parser uses strategies to minimize memory footprint of the final parser in ways appropriate for computers with kilobytes of memory, but inappropriate to modern computers with gigabytes of memory.

The translator reads in the .c and .h files generated by the Lemon Parser and writes a C# file that defined a partial class called Parser as follows:

1. The Lemon parser allows the user to modify the parser’s actions by using the #define preprocessor directive to add or remove lines of source code. For example, #define NDEBUG turns off tracing by deleting tracing code and omitting a #define turns on tracing by not omitting the code. The C# translator declares these as public properties and use if else statements to execute or skip code. The user can modify these properties as needed.
2. The Lemon parser #define preprocessor directives to define constants. The translator declares these as private read-only variables.
3. Parsing tables are converted to C# format.
4. A method ExecuteReductions is created to implement reductions.

The remainder of the definition of the Parser class is in the file Parser.cs.

**Workflow**

The workflow is as follows:

1. Create a .y file to specify the language grammar.
2. Use the Lemon Parser to use the .y file to create a .c file and .h file.
3. Use LemonTranslateCS to program to convert the .c and .h files to create a C# partial class definition.
4. Creates a file called YYMINORTYPE.cs that defines a class of the same name to encapsulate the data of the target language.
5. Use the partial class definition from step 3 with YYMINORTYPE.cs and Parser.cs to create an assembly or executable.
6. To parse, define a new Parser object. For example:

Parser p = new Parser().

1. Repeatedly call the parser with tokens and their values (as XXMINORTYPE objects.

p(tokenID,value0

1. When done, call:

p.End();

Unrecoverable syntax errors throw an ApplicationException.

To reset the parser call:

p.Reset();

There follow following functions are not necessary with the C# parser and are not supported:

void \*pParser = ParseAlloc( malloc );

ParseFree(pParser, free);

ParseTrace(FILE \*stream, char \*zPrefix);

**Unsupported Lemon Parser Directives**

* %code
* %default\_destructor
* %default\_type
* %destructor
* %else
* %endif
* %extra\_argument
* %if
* %ifdef
* %ifndef
* %include
* %name
* %stack\_overflow
* %stack\_size
* %token\_class
* %token\_destructor
* %token\_prefix
* %token\_type
* %type
* %parse\_accept
* %parse\_failure
* %syntax\_error

**Unsupported Lemon #Defines**

* Parse\_ENGINEALWAYSONSTACKcase
* ParseARG\_FETCH
* ParseARG\_PARAM
* ParseARG\_PDECL
* ParseARG\_SDECL
* ParseARG\_STORE
* ParseCTX\_FETCH
* ParseCTX\_PARAM
* ParseCTX\_
* ParseCTX\_SDECL
* ParseCTX\_STORE
* ParseTOKENTYPE
* TOKEN
* YYACTIONTYPE
* YYCODETYPE
* YYMALLOCARGTYPE
* YYMINORTYPE
* YYPARSEFREENEVERNULL
* YYTRACKMAXSTACKDEPTH