CSC435/535: Assignment 1
(Due: 11:55pm, 30 May 2014)

Introduction

This assignment asks you to complete the lexical analysis and parsing phases of the front-end for the CFlat (Cb) compiler.

In addition, you should prepare one more test program in the Cb language which will run and do something interesting. Your test program should exercise some Cb language features which are not used in the sample program `CbExample.cs` which is provided with this assignment.

You are provided with an incomplete parser specification in the `CbParser.y` file. (In particular, the grammar rules in that file do not allow multiple classes in the program or multiple methods in a class.)

Assignment Description

1. Your Cb test program should compile and run with the regular C# compiler.
2. You are provided with an incomplete grammar for the Cb language in a form which is accepted by the `gppg` parser generator. This is the file named `CbParser.y`. If the command
   ```
   gppg /gplex /conflicts CbParser.y > CbParser.cs
   ```
   is executed, a parser module file named `CbParser.cs` and coded in C# is generated.
3. Your task is to provide a complete lexical analyzer (*aka* scanner or lexer) which works with the generated parser. The `gplex` tool should be used to generate that lexer. The C# file which contains the lexer should be named `CbLexer.cs`.
4. You also need to provide a main program which invokes the lexer and parser on a Cb source file. The main program should be implemented as a file named `cbc.cs` which contains a class with a name of your choosing. That class must contain a static method named `Main`. The `Main` method accepts command line options and exactly one filename argument. An example invocation of the lexer+parser showing every option is as follows:
   ```
   cbc -tokens -debug sampleprog.cs
   ```
   where `cbc` is the name of the program being invoked (taken from the filename `cbc.cs`) and `sampleprog.cs` is a textfile holding a sample Cb program.
   The `-tokens` option causes the lexer to generate a listing of all the tokens which are encountered in the Cb source file. The listing should be written to a new file named `tokens.txt`, with one token per line. A few lines of the listing might be
   ```
   Token.Kwd_class
   Token.Ident, text = "Foo"
   Token "{"
   ```
   where the representation of each token (e.g. the name `Kwd_class`) is taken from the names used for these token types in `CbParser.y` (see the `Tokens` enum type in the `CbParser.cs` file).
   The `-debug` option sets a flag which enables your own debugging output. You will find this option to be very useful when developing the later stages of the compiler. What you output is completely up to you. (Indeed you can output nothing at all.)
   When a Cb source file is successfully parsed, there should be a succinct output message similar to the following:

```
        237 lines from file cbtest.cs were parsed successfully
```

If a lexical error or a syntactic error is discovered, an error message which specifies the line number and the name of the source file should be generated. The default message produced by a gplex scanner or gppg parser is quite acceptable as long as the message includes the location of the error. (The column number within a line need not be provided.)

Advice

Your tasks can be performed in this order:

1. Add `%token` declarations to the `CbParser.y` file for all keywords mentioned in the Cb Language Specification document but which are missing.
2. Implement a scanner which recognizes all the lexical elements listed in the Cb Language Specification.
3. Compare the EBNF grammar against the grammar implemented by the `CbParser.y` file and add any missing Cb language constructs to the `CbParser.y` file. Generate the parser code.
4. Write the `cbc.cs` file so that it calls the parser on the specified test file. Compile the three files together.
5. Construct a small trivial Cb test file and check to see if your scanner+parser combination successfully recognizes it. If there are problems recognizing the input, fix the problems and try again. Otherwise, expand your test file to use more features of the Cb language and repeat.

The Provided Materials

- The conneX website holds the most recent versions of the `gplex` and `gppg` programs, along with full documentation for them.
- The file `CbLangSpec.pdf` holds a succinct specification of the Cb language. Note that Cb is intended to be a proper subset of C#. If the meaning of any construct is not explained or unclear, just assume the usual C# meaning.
- The file `CbExample.cs` holds a sample Cb program.

Submission Requirements

1. You must provide four files. The three files which implement the lexer+parser should be named `CbLexer.lex`, `CbParser.y` and `cbc.cs`. The sample Cb program can be named anything as long as the filename makes it obvious that this is a test program or sample program, and has the suffix "`.cs`". A name beginning with the letters "test" would be appropriate.
2. Important: do *not* submit any files generated by gplex or gppg.
3. You must combine all your files into a single compressed archive file. The only accepted formats for the archive file are as a zipfile (and the filename must have a "`.zip`" suffix) or as a gzipped Unix tar file (and the filename must have a "`.tgz`" suffix).
4. The project is to be completed in teams of either 2 or 3 persons. The ideal size is 2 people. All team members *must* participate. To encourage and to reward active participation, you can expect the midterm test to contain at least one question on the minutiae of Cb semantics. Team memberships can be changed after each assignment,