CSC435/535: Assignment 2
(Due: 11:55pm, 15 June 2013)

Introduction

This assignment asks you to attach tree building actions to the rules of the Cb grammar so that an abstract syntax tree (AST) is created for any Cb program.

Two AST visitor classes, implemented using the Visitor design pattern, are to be included in the `cbc` program. One visitor constructs a top-level dictionary structure containing some details for namespaces and classes. The other visitor prints the entire contents of the AST and the code for this visitor is provided as an example.

Assignment Description

The introduction said it all. The AST must be implemented using the classes and node tags which have been provided in the `CbAST.cs` file (see below).

You may want to adjust the grammar rules in order to simplify the task of building the AST. This is permitted provided that the language accepted by the parser is not impacted. Your parser can accept a superset of the Cb language described in the Cb specification as long as the additional features are also accepted by the C# compiler.

Building the AST

The `AST-DataStructure` document explains the AST structure in some detail. The names used for node tags correspond very closely to names of non-terminal symbols in the grammar.

A few semantic actions for creating AST nodes are provided in the `CbParser.y` file and can be used as examples to imitate. (Creating a k-ary node is not obvious, so it's best to just imitate the provided examples.)

You will need to add semantic actions to the ends of (nearly all) the production rules in the grammar, and these actions will construct new nodes, and pass references to nodes to constructors, as necessary.

Note that every AST node is supposed to contain the corresponding line number in the source code. When a construct (say a while loop) spans two or more lines, we probably want to associate the starting line number with the node for that construct rather than the ending line number. However if we look up the current line number when we allocate the AST node, we will usually be looking it up when the parser has reached the end of the construct. (The exception is for k-ary nodes which are normally allocated at the start of a construct.) A better line number can be obtained by copying it from the first subtree of this new node – examples of this can be seen in the supplied code.

Building the TopLevel Namespace Dictonary

You will need to program a new subclass of `Visitor`. (See `CbPrVisitor.cs` for an example.) The visitor methods have an extra parameter of type object which can be used to pass information around during the traversal. If you do not need to use this parameter, just pass `null` as the value rather than create new `Visit` and `Accept` methods.

When invoked from the code in the `cbc.cs` file, this visitor will traverse the AST and perform the following actions:

- If it encounters a "`using System`" declararation, it must copy all classes in the `System` namespace (this is Cb's stripped down version of `System`, not the real C# one) to the top level, so that a method invocation such as
  ```
  Console.WriteLine("Hello");
  ```
  can be used in addition to the long-winded but correct
  ```
  System.Console.WriteLine("Hello");
  ```
  I.e., the type description of the `Console` class (and a few other classes) must be copied from the `System` namespace to the top-level namespace.
- If it encounters a declaration of a class, a type descriptor for that class (see the `CbType.cs` file) must be created and added to the top-level namespace.
- If it encounters declarations of members of a class (these can be constants, fields, and methods), it must create an appropriate descriptor for that member and add it to the class's type descriptor.
- The only semantic checking that need be performed during the traversal concerns checking for duplicate declarations of names. In Cb, we cannot declare two classes with the same name and a class cannot contain two members with the same name.
  The `AddMember` methods of the `NameSpace` and `CbClass` types return a result to indicate whether the name was a duplicate. If a duplicate is found, an error message with an appropriate line number should be generated.

The Provided Materials

- You are provided with everything needed to start this assignment. If you wish, you may discard your solution to Assignment 1 and start over with the files provided on the conneX website. Alternatively, you may choose to keep working with your original files (and copy corrections and improvements from the supplied files into your own code).
- The supplied source code files are listed in the table below.

File	Description
`cbc.cs`	The main program which invokes everything else.
`CbLexer.lex`	The lexical specification file to be processed by gplex
`CbParser.y`	The grammar file to be processed by gppg
`CbAST.c`	The classes used for building the AST
`CbVisitor.cs`	The parent class for the Visitor pattern
`CbPrVisitor.cs`	A visitor for printing the AST
`CbType.cs`	Classes used for describing Cb datatypes
`CbTopLevel.cs`	Classes used for contents of Cb namespaces
`AST-DataStructure.pdf`	Explanation of the AST structure

- For Windows users, three batch command files are provided:

`rungplex.bat`	Runs gplex on `CbLexer.lex`
`rungppg.bat`	Runs gppg on `CbParser.y`
`build.bat`	Builds `cbc.exe` from all the C# source files

Linux or Mac OS X users can copy the commands into a Makefile or into shell scripts.

Submission Requirements

1. You must provide exactly one file. It must be a zip file or gzipped tar file which contains the source code files `cbc.cs CbLexer.lex CbParser.y CbAST.c CbVisitor.cs CbPrVisitor.cs CbType.cs CbTopLevel.cs`.
   If you added more C# files to your project, include those too.
2. Also include a file named `README.txt` which identifies the team members. If you have any comments you want to share about problems with the assignment, this is an appropriate place to supply the comments.
3. Important: do *not* include any files generated by `gplex` or `gppg` in your submission. (You of course must not edit the C# code output by these two tools.)
4. Again, the project is to be completed in teams of either 2 (or possibly 3) persons. The ideal size is 2 people. The teams do not have to contain the same members as for Assignment 1.