Instructions

This project was written in C# and requires the *.Net Framework 4.7* to be installed
https://www.microsoft.com/en-us/download/details.aspx?id=55170

It can be compiled using *Visual Studio Community 2017*
https://www.visualstudio.com/downloads/

To compile the project, open the solution file (.sln) using Visual Studio. To run the program under debug mode, hit F5.

The executable can be found in Assignment4/Assignment4/bin/debug/

This application also generates a comprehensive .html report file for each individual parse with varying information about the parse. It uses Javascript and CSS libraries and was **only** tested on the latest version of Firefox for Windows 10. Support for situations in different conditions may vary.

PLEASE NOTE: When performing a GET operation:

FOR INTEGERS – Enter the FULL integer before hitting enter.

FOR FLOATS – Floats take in TWO INTEGERS. One is the REAL portion, the other is the EXPONENT. First enter the FULL REAL portion of the floating point number with 9 digits before hitting enter. Then, enter the FULL EXPONENT of the floating point number before hitting enter.

Brief Overview

Code Generation

MemorySizeVisitor.cs	-	The first visitor: determines the size of all types and variables within the proram.
StackIncreaserVisitor.cs	-	The second visitor: uses the memory sizes to designate an offset space on the stack for the variables within the program, as well as sub-calculation variables.
MoonVisitor.cs	-	The third visitor: uses the offsets and the memory sizes to generate MoonCode for the program.
InstructionStream.cs	-	Manages and contains the list of MoonCode instructions for the program.
Report.cs	-	Generates report sections relating to the code generation,
Puts.m	-	Contains hand-written input and output libraries for printing to the screen, and retrieving values from the user.

Lexical Analyzer

Scanner.cs	-	Scans through the characters of a file.
Language.cs	-	Contains constant information about the language.
Token.cs	-	The data structure of a token. Also contains conversion of a token to AtoCC format.
TokenStream.cs	-	A queue data structure simulating a stream of tokens.
TokenTypes.cs	-	An enumeration of all the different token types.
Tokenizer.cs	-	Handles parsing through a file and converting the character stream into a TokenStream.
Report.cs	-	Generates report sections relating to the lexical analysis,
Extensions.cs	-	Contains extension methods for determining whether or not a character falls under a certain set of characters.

Synactic Analyzer

Extensions.cs	-	Contains extension methods for identifying tokens in a first/follow set, and joining lists.
Report.cs	-	Generates sections for a report that contains information about the parse done by the syntactic analyzer.
Parser.cs	-	Parses a token stream and returns an abstract syntax tree.
Deriver.cs	-	Contains logic for applying rules to establish the program derivation.
Nodes\	-	Contains the node structures for the abstract syntax tree.
NonTerminals\	-	Contains the logic for each non-terminal in the LL(1) grammar. Each non-terminal applies error detection and recovery, AST node creation, and applies derivation rules.
Classification.cs	-	An enumeration containing the different types of Symbol Table entries.
IVisitable	-	An interface for allowing Nodes of the AST to be visited.
Visitor.cs	-	An abstract class defining PreVisit and Visit methods to be used in the Semantical Analysis.
TableEntry.cs	-	A data structure defining an entry in a Symbol Table.
SymbolTable.cs	-	A data structure defining a Symbol Table.

Semantical Analyzer

SymbolTableVisitor.cs	-	The first pass of the Semantical Analysis. Constructs the basic symbol tables of the program.
FunctionLinkerVisitor.cs	-	The second pass of the Semantical Analysis. Links function implementation to their declaration in their respective classes.
InheritanceVisitor.cs	-	The third pass of the Semantical Analysis. Migrates class fields and functions to their inheriting classes.

TypeVisitor.cs	-	The fourth pass of the Semantical Analysis. Applies type checking, ensures variable and function existence, enforces the correct number of parameters, indices, and type of indices.
LanguageSemantics.cs	-	Applies all the visitors on a given AST.
Report.cs	-	Generates report sections relating to the semantical analysis,
Extensions.cs	-	Contains extension methods for repeating a string numerous times.

Errors

Error.cs	-	A data structure containing information about an error.
ErrorManager.cs	-	Collects and stores errors.
WarningManager.cs	-	Collects and stores warnings.

Report Generator

Report.cs	-	Contains all the sections of the report, and generates a table of contents for all the sections. Adds all necessary Javascript and CSS libraries to the page.
Section.cs	-	Contains all the content for a section of a report.
IReportable.cs	-	An interface intended to be used to generate sections for a report.

Assignment4

Program.cs	-	Contains the main method for the Assignment 4.

Test Cases

Tests can all be found in the Tests folder, and their expected output can be found in their corresponding .html report file.

PLEASE NOTE: The derivation of a program is only presented in the report if the syntax is of a valid program.
The AToCC Output Stream is only presented if the program is a valid program.
The Abstract Syntax Tree Traversal is only presented if the program is a valid program.
The Abstract Syntax Tree is only presented if the program is a valid program.
The Symbol Tables can be found at the bottom of each .html file, regardless of whether or not the program is valid.

Files of the .txt format are the source files.

Files of the .xml format are the resulting AST's serialized to XML. They can be viewed in a visual tree in their corresponding .html report file.

Files of the .html format are reports generated by the parser displaying all relevant information about the parse.

There is a .css and .js file in the test folder as well. This is needed for viewing the .html reports.

The code generated by the program can be found in the same filename with a .m extension.

Tools

I used C# and Visual Studio because of my familiarity and comfort with the language and the IDE.

For the purposes of this and later assignments, I wrote and used a library for generating .html report files (included in the source files).

I used Bootstrap for formatting the .html reports because of its widespread usage and cross-browser support.

I used Xml2Tree for representing the Abstract Syntax Tree in its respective .html report because it was able to represent a collapsible DAG within the browser from an XML file. I was able to serialize the AST to XML and then include a function call to Xml2Tree to represent it in the browser.