Compiler Construction
Project 1 Documentation

Expression. When assigning operator precedence and associativity I take into account some modern browser implementations like that of Mozilla Firefox and Chrome, so my implementation will be slightly different (and they're all listed below):

1. Ternary operator "a ? b : c" is implemented as right associative. This makes more sense if we take a look at expressions of the form "a ? b : c ? d : e" which is intuitively interpreted as "If a holds then this expression evaluates to b, otherwise if c holds then it evaluates to d, or e if c doesn't holds." Thus the expression should be grouped as "a ? b : (c ? d : e)" which implies the right associativity of the ternary operator.
2. Member access operator ":" and function call operator "()" are assigned the same precedence to allow two different types of expressions to be unambiguous without parentheses.

    i. "foo().bar", that is, foo is of type "function" that returns an object that has a member "bar". This would've been invalid if we had assigned "." operator higher precedence. Same goes to line 7 of the given sample "Fastfib.ssc".

    ii. "bar.foo()", that is, bar is an object that has a member "foo" which is callable.
3. Three assignment operators are added: "-=", "*=" and "/=".
4. Comma operator is added to have the lowest precedence and is left associative. It evaluates a sequence of expressions and returns the value of the last expression.
5. Interface variable initializer production "$E \rightarrow$ {(Id:E)*}" is removed. To enable the same feature, "Object Literal" is introduced.
6. "Object literal" initializes an object and adds members to that object according to all the name-value pairs defined in the curly brackets. For example, "{x : 4, y : 5,}" returns an object which

has two members **x** and bf y with values 4 and 5. Note that an optional trailing comma is allowed by the grammar.
If we are to implement a strong typing language in the future, additional type checking can be performed before assigning an object literal to an interface variable. The grammar also allows nested object literal and passing an object literal to a function call, which saves the programmer a lot of pain from defining temporary variables.

Declaration. Some extensions are made to create a more programmer-friendly language.

1. Variable declaration statement is extended to allow defining multiple variables within one statement (definitions separated by comma).
2. Anonymous function literal is introduced yet disabled (please refer to line 44 and sort **AnonFunc**). This introduction is just for fun since the real implementation requires function closure feature which requires the compiler to embed a runtime to handle dynamic memory allocation and garbage collection. I'm not sure if the ARM32 machine code is sufficiant to support this feature (and if I have time to implement that.)
3. Originally interface declaration and function declaration were both treated as statements. This modification was to enable declarations within function body. If we were to implement this modification, we would have to define the scoping of those nested declarations and to implement a runtime which supports function closure. Attached file "promise.ssc" can be parsed by the alternative parser compiled from "Pr1Yuhan.hx.original".

Program The original definition of the main sort $Program \rightarrow StatementSeq$ which is commented out, allows declarations and statements to be interweaved (It also accepts empty input as of no declarations nor statements).
Current version "Pr1Yuhan.hx" does conform to the definition which requires at least one statement.