Project Milestone 2 part A

Abhi Agarwal

My project two documentation is organized in a way where I break down the details I'm going to present and explain them in detail, and then I construct the table. I have also attached definitions of word I'm using at the bottom to reduce ambiguity and misunderstandings.

2.1 - Assignment compatible

Definition: Assignment compatible applies to us here because we know that: one type of a value (T_1) is assignment compatible with a second type of a value (T_2) if the value of the first type can be assigned to the value of the second type (if $type_2\ T_2 = T_1$ is possible).
Analysis: To me (my assumption is that) `any` means that we're able to convert between types very explicitly without using a cast, and we're able to assign certain values regardless of the type. Moreover, the every type is assignment compatible with itself is fairly straight forward, and the standard case.

2.2 - Literals

Identifier

Definition: Identifier are tokens that start with a letter, $, or _, followed by more of the same as well as digits, except that keywords (literal tokens used by the grammar) are not permitted as identifiers.
Scope: Here we're looking at values within the current scope.
Analysis: Identifier literals must exist within the scope where it is declared, and has to have the correct type. We have to do a type check to see if the assignment is the same as the type it was declared as.

Integer

Definition: Integer tokens are sequences of digits.
Scope: Here we're looking at values within the current scope.
Analysis: We've to check if they've the type int to classify them as Integers. This is just mechanical as we've to simply check if their type matches the type `int`.

String

Definition: String tokens are sequences of characters enclosed in either ' (single quote) or " (double quote).
Scope: Here we're looking at values within the current scope.
Analysis: We've to check if they've the type string to classify them as `string`.

object

Definition: object literal $\{k_1$:Literal1,... ,k_n:Literaln$\}$ for n $\geq$ 1 with each key k_i an Identifier (also called "field name").
Scope: Here we've to create a new nested scope as we enter a new block denoted by the curly braces ({}).
Analysis: We must check if m_k for each k to see if it is not already within the current scope, and if it is then we have to generate an error as they must be distinct.
We also have to check from the scope of the class to make sure the m_k exists in the class type.
We also have to check for the type of l_k, for a given k, and if it has a type which is assignment compatible to the type declared for the member then it's been validated.

2.3 - l-value

Definition: What is allowed to have a left side of assignments. Any Expression that has an equal symbol, or is being set to something, has to have one of these forms on the left side of the assignment: an Identifier or a member access form (i.e.: $e.m$)
Scope: Here we're looking at values within the current scope.
Analysis: We have to check whether the lexeme on the left site is either an Identifier or an instance of a class variable for which the element it is trying to access exists, and if either of these conditions are not present then we return an error.

2.4 - Expressions

1. this

Definition: Referencing a variable defined locally within the scope of a class, and therefore this may only be used within the block of a class.
Scope: The current scope has to be within the global scope of a class declaration.
Analysis:

2. Referencing members within the type class

Definition: For $E.m$ with m a member name, E must be of a declared class type C that has the member m;
Scope: Here we're looking at values within the current scope, but we're also looking at the scope variables of the class.
Analysis:

3. Function calls

Definition:
Scope:
Analysis:

4. $!E$ and E

Definition: The operations listed above must be of type boolean as is the result.
Scope: Here we're looking at values within the current scope.
Analysis:

5, 6. E, $-E$, $+E$, E_1*E_2, E_1/E_2, $E_1\%E_2$, and E_1-E_2

Definition: The operations listed above must have type int as does the result.
Scope: Here we're looking at values within the current scope.
Analysis:

7. E_1+E_2

Definition: The operations listed above must have type int or string; the result is type int if both have type int and string otherwise.
Scope: Here we're looking at values within the current scope.
Analysis:

8. $E_1 < E_2$, $E_1 \leq E_2$, $E_1 > E_2$, and $E_1 \geq E_2$

Definition: The operations listed above must have the same type, which should be either int or string; the result is of type boolean.
Scope: Here we're looking at values within the current scope.
Analysis:

9. $E_1 == E_2$ **and** $E_1 != E_2$

Definition: The operations listed above must have the same type; the result is of type boolean.
Scope: Here we're looking at values within the current scope.
Analysis:

10. $E_1 \&\& E_2$ **and** $E_1 || E_2$

Definition: The operations listed above must have type boolean as will the result
Scope: Here we're looking at values within the current scope.
Analysis:

11. $E_1 = E_2$

Definition: E_1 must be an l-value such that the type of E_2 is assignment compatible to the type of E_1; the result type is the type of E_2.
Scope: Here we're looking at values within the current scope.
Analysis:

12. $E_1 += E_2$

Definition: E_1 must be an l-value (Rule 11) of either type int or string (Rule 7).
Scope: Here we're looking at values within the current scope.
Analysis:

13. E_1, E_2

Definition: The operations listed above must be inside function call argument lists.
Scope: Here we're looking at values within the current scope.
Analysis:

2.5 - Statements

1. Block statement scope

2. New variable declaration

3. Tests on while and if statements

2.6 - Class

1. Scope

Definition: Class declaration class C $\{m_1...m_k\}$ declares C as a type in the top scope.
Scope: Here we're looking at values within the top scope.
Analysis:

2. Field member declaration

Definition:
Scope:
Analysis:

3. Method member declaration

Definition:
Scope:
Analysis:

2.7 - Function declaration

2.8 - Deceleration precedence

2.8 - Deceleration precedence

2.9 - Operations on strings

Syntax Directed Definition Table

Production	Semantic Rules
item 1	item 2

Definitions

Skip over this section, but here I've defined some things I think are important, and I wanted to have the definitions here to reduce ambiguity and misunderstandings.
Scope: Tied to program "blocks" (lexical) vs runtime stack.
Global scope: Scope that surrounds a block where the block can be a function, variable, object, or class.
Top scope: The scope of the whole program that includes all declarations of classes, functions, and global variables that are declared in the main script.
Current scope: For a given block we're currently in, all the local functions/variables/-classes we've encountered.
New scope: We're going into a new program block, and therefore we declare a new nested scope including values from the global scope.