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 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

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.
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 scope with a with inherited values from the global scope.