P15/1033/2016 – ADUMA TEVIN JOEL

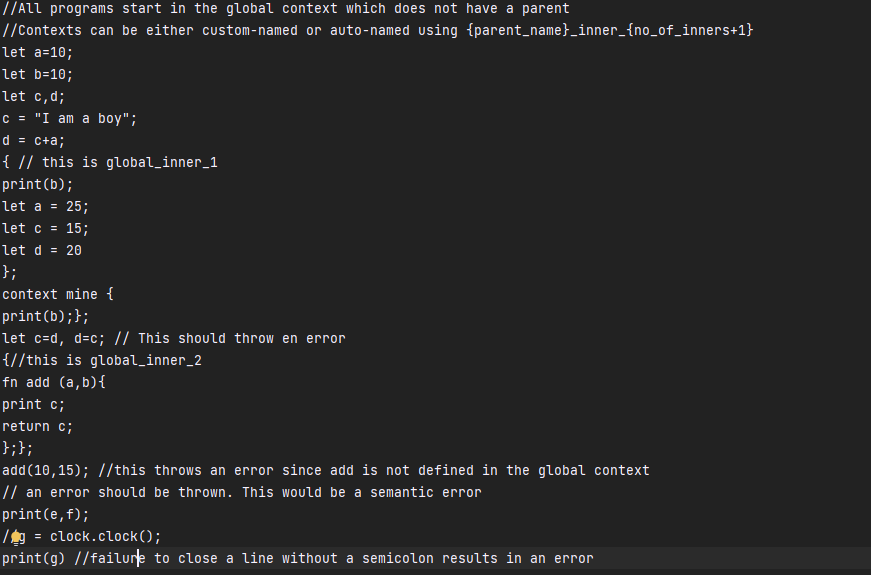
P15/1724/2019 – ONYANGO BENTLEY PHEMBER

P15/37390/2016 – WAMAE JOSEPH WANYOIKE

P15/2142/2021– GREGORY NYAMAO KAISER

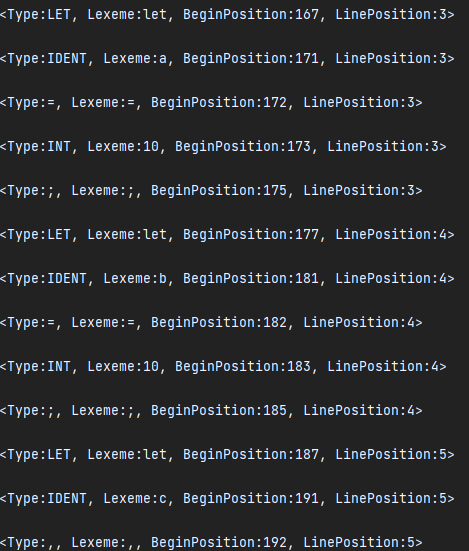
|  |  |  |
| --- | --- | --- |
| Task/Function Module | Estimated Effort (Days and Duration) - 14 | Responsible Group Member |
| Symbol Creation | 2 days (April 30th - May 2nd) | Wamae |
| Symbol Type Setting | 1 day (May 2nd - May 3rd) | Wamae |
| Symbol Representation | 2 days (May 3rd - May 5th) | Wamae |
| Context Management | 5 days (April 30th - May 4th) | Tevin |
| Symbol Definition | 1 day (May 2nd - May 3rd) | Greg |
| Symbol Lookup | 1 day (May 3rd - May 4th) | Greg |
| Table Representation | 3 days (May 4th - May 7th) | Tevin |
| Token Stream Consumption | 2 days (May 6th - May 8th) | Tevin |
| Statement Parsing | 3 days (May 3rd - May 7th) | Greg |
| Expression Parsing | 3 days (May 7th - May 10th) | Bentley |
| Error Handling | 1 days (May 10th - May 11th) | Bentley |

We created similar parsers that utilize both top-down and recursive-descent methods. The former was done in Python and the latter was built in Python.

**Source Code**

* **Syntax Errors**
  + The code contains several syntax errors, such as missing semicolons and incorrect variable declarations (e.g., let =10; should be let a = 10;).
* **Semantic Errors**
  + There are semantic errors indicated by comments, such as the use of undeclared functions (e.g., add(10,15); where add is not defined in the global context).
* **Context Management**
  + The code demonstrates the use of contexts, with comments explaining the naming conventions for custom and auto-named contexts.
* **Function Definitions**
  + A function add is defined with parameters a and b, but it is incorrectly referenced outside its scope, leading to an error.
* **Error Handling**
  + The code includes comments where errors are expected, providing insight into potential issues with undeclared identifiers and missing semicolons.

**Below is the token list**



1. **Symbol Creation**

Initialises a Symbol object with attributes such as **node, type\_, name, line\_declared, line\_referenced, context\_level, value,** and **errors.**

The node attribute represents a node in the AST, and the context\_level attribute indicates the level of context in which the symbol is defined.

Calls the **create\_symbol** method to create the symbol based on the provided node.

**Method ‘create\_symbol’:**

This method is responsible for creating a symbol based on the provided node.

It checks if the node is an instance of Node or SymbolTable, and sets the attributes of the symbol accordingly.

Determines the ***type\_ of*** the symbol based on the type of the node.

Handles different cases such as function literals, context, and symbol tables.

If the value of the node is empty, it handles such situations.

Returns True after creating the symbol.

1. **Symbol Type Setting**

In the **Symbol** class, the **set\_type** method is responsible for determining the type of the symbol based on the type of the associated AST node

.

**Method ‘set\_type’**

Determines the type of the symbol based on the type of the node.

Sets the ***type\_ attribute*** of the symbol accordingly.

Handles special cases such as ReturnStatementNode and StatementListNode.

1. **Checking Node Type**

* The method first checks if the associated AST node ***(self.node)*** has a ***type*** attribute. If it does, it assigns the value of that attribute to the ***type\_*** attribute of the ***Symbol*** object.
* This allows the type of the symbol to be explicitly set based on the type attribute of the AST node.

1. **Special Cases Handling**

* If the AST node's type matches **ReturnStatementNode**, the ***type\_*** attribute of the ***Symbol*** object is set to '***FUNCTION DEFINITION'***. This indicates that the symbol represents a function definition.
* If the AST node's type matches ***StatementListNode***, the ***type\_*** attribute of the ***Symbol*** object is set to '***CONTEXT***'. This indicates that the symbol represents a context or block of statements within the code.

1. **Returning the Type**

After determining the type of the symbol, the method returns the type\_ attribute.

This allows the caller to access the type of the symbol and use it for further processing or analysis.

**3.Symbol Representation**

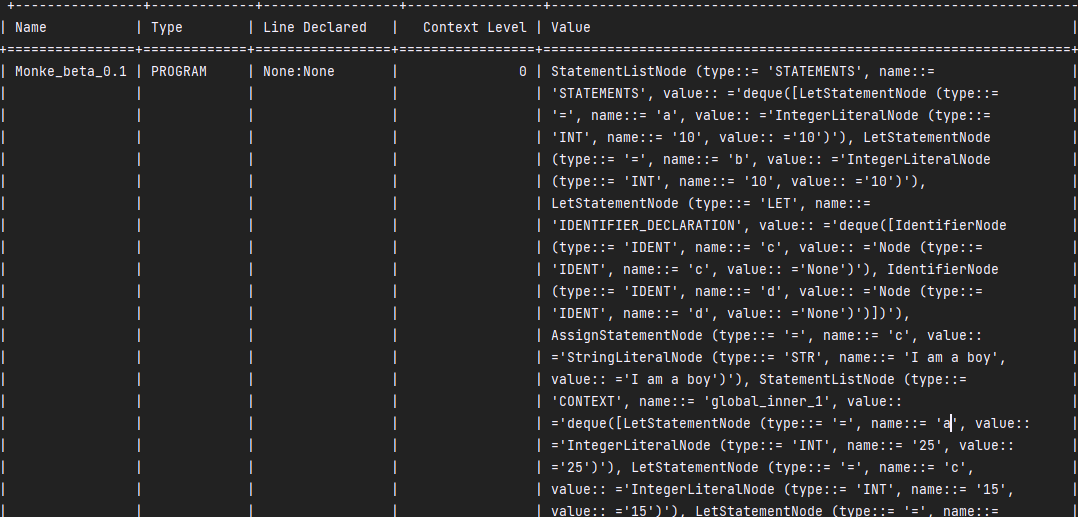
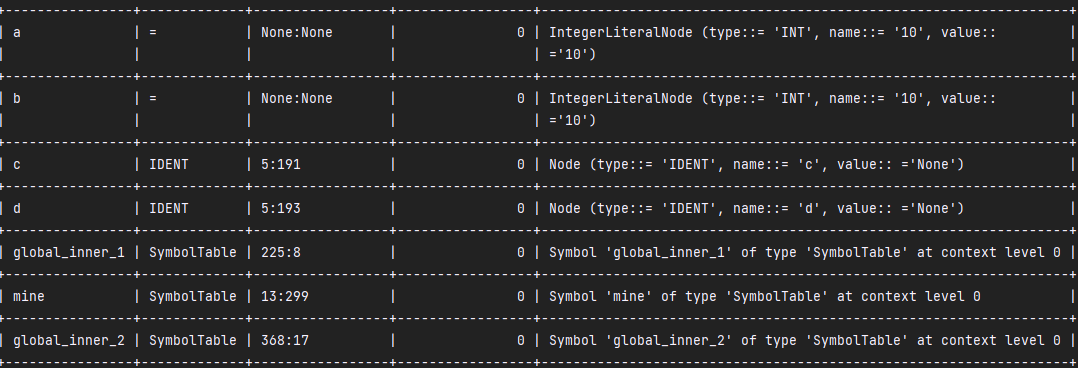
The **Symbol** class provides two methods for representing a **Symbol** object: **\_\_repr\_\_** and **\_\_str\_\_**

***\_\_repr\_\_*** and ***\_\_str\_\_*** methods:

The ***\_\_repr\_\_*** method returns a formatted string containing the name, type, and context level of the ***Symbol*** object. For example, if ***name='x', type\_='variable',*** and ***context\_level=1***, the representation would be ***"Symbol(name='x'***, ***type\_='variable'***, ***context\_level=1)***

***\_\_repr\_\_*** provides a string representation that can be used to recreate the object, while ***\_\_str\_\_*** provides a more human-readable representation.

**SymbolTable Class Report**

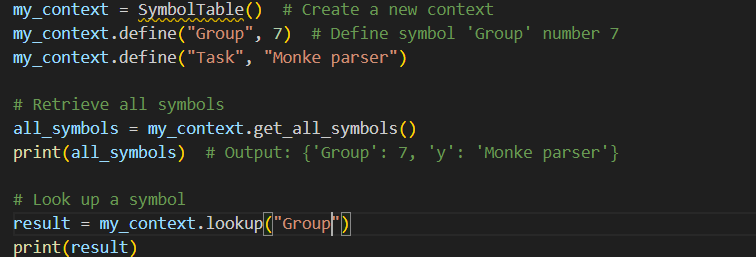
1. **Table Representation**
   * **Description**: The SymbolTable class provides a formatted view of the symbol table for debugging purposes.
   * **Necessity**: This representation assists developers in visualizing the current state of the symbol table during program execution.
   * **Responsible Member**: TEVIN
2. **Token Stream Consumption**
   * **Description**: The SymbolTable class manages the flow of tokens being consumed by the parser.
   * **Necessity**: Core to the parsing process, it allows the parser to read tokens sequentially from the input stream.
   * **Responsible Member**: TEVIN
3. **Context Management**
   * **Description**: The SymbolTable class handles entering and exiting of contexts (scopes) and nesting of scopes.
   * **Necessity**: Context management is fundamental for scope resolution, variable visibility, and maintaining the correct working context during parsing.
   * **Responsible Member**: TEVIN

Parser Implementation for mini-grammar.

**Symbol Definition**

**Define:** This symbol represents the function that defines a new symbol in the current context. It takes two parameters: name (the name of the symbol) and symbol (the value associated with the symbol).

get\_all\_symbols: This symbol represents the function that retrieves all symbols from the current context and its parent contexts. It returns a dictionary containing all symbols.

lookup: This symbol represents the function that looks up a symbol by its name in the current context and its parent contexts. It returns the value associated with the symbol or None if not found.  
for example   
Functionality:

Retrieves the current context where the symbol will be defined.

Checks if there is an active context available.

Verifies if the symbol with the given name already exists in the current context.

If the symbol does not exist, adds the symbol to the current context under the given name.

**Statement Parsing**

**1. define(self, name: str, symbol: Symbol):**

* **Functionality:** This method is used to define a symbol in the current context of the symbol table.
* **Parameters:**
  + **name**: A string representing the identifier of the symbol to be defined.
  + **symbol**: An object of type **Symbol** representing the properties and metadata associated with the symbol.
* **Process:**
  + It retrieves the current context from the symbol table.
  + If there is no active context (not represented as a dictionary), it raises a **NameError**.
  + Checks if the symbol with the given name already exists in the current context. If it does, it raises a **NameError**.
  + Otherwise, it adds the symbol to the current context under the given name.
* **Return:** None

**2. get\_all\_symbols(self):**

* **Functionality:** This method retrieves all symbols defined in the symbol table.
* **Process:**
  + Iterates over all contexts in the symbol table.
  + If a context is represented as a dictionary, it updates the **all\_symbols** dictionary with the symbols from that context.
  + If a context is an instance of **SymbolTable**, it adds a representation of that context to the **all\_symbols** dictionary.
* **Return:** A dictionary containing all symbols in the symbol table.

**3. lookup(self, name):**

* **Functionality:** This method is used to look up a symbol by its name in the symbol table.
* **Parameter:**
  + **name**: A string representing the name of the symbol to be looked up.
* **Process:**
  + Starts with the current context and traverses up through the parent contexts.
  + Searches for the symbol with the given name in each context.
  + If the symbol is found, it returns the corresponding **Symbol** object.
  + If the symbol is not found in any context, it returns **None**.
* **Return:** The **Symbol** object if found, otherwise **None**.

**Symbol Lookup**

1. **Start from Current Context**: The lookup process begins from the current context within the symbol table.
2. **Traverse through Parent Contexts**: If the symbol is not found in the current context, the method traverses up through the parent contexts recursively until either the symbol is found or there are no more parent contexts to search.
3. **Search for Symbol**: Within each context, the method checks if the symbol with the given name exists in the context's symbol table. If the symbol is found, the corresponding **Symbol** object is returned.
4. **Return Result**: If the symbol is found, the method returns the **Symbol** object representing the found symbol. If the symbol is not found in any context, the method returns **none**, indicating that the symbol is not defined in the symbol table.

Here's a breakdown of the method's functionality:

* **Parameter**:
  + **name**: This is the name of the symbol to be looked up.
* **Process**:
  + The method starts with the current context and traverses up through the parent contexts until the symbol is found or the search reaches the root of the symbol table.
  + Within each context, it checks if the symbol with the given name exists in the context's symbol table.
  + If the symbol is found, the corresponding **Symbol** object is returned.
  + If the symbol is not found in any context, **None** is returned.
* **Return**:
  + If the symbol is found, the method returns the **Symbol** object representing the found symbol.
  + If the symbol is not found, the method returns **None**.

**Parsing and Error Handling**



