Compilers Project

Decaf compiler

Mohit Sharma
201505508

Phase 3:

Our objective for phase 3 is to build the Abstract Syntax Tree (AST) for the parsed program. Then to traverse the tree to produce nested XML output which represents the structure of the program.

Input:

```
class Program
{
	boolean field;
	int x[10];
	int foo(int y)
	{
	}
}
```

Output:

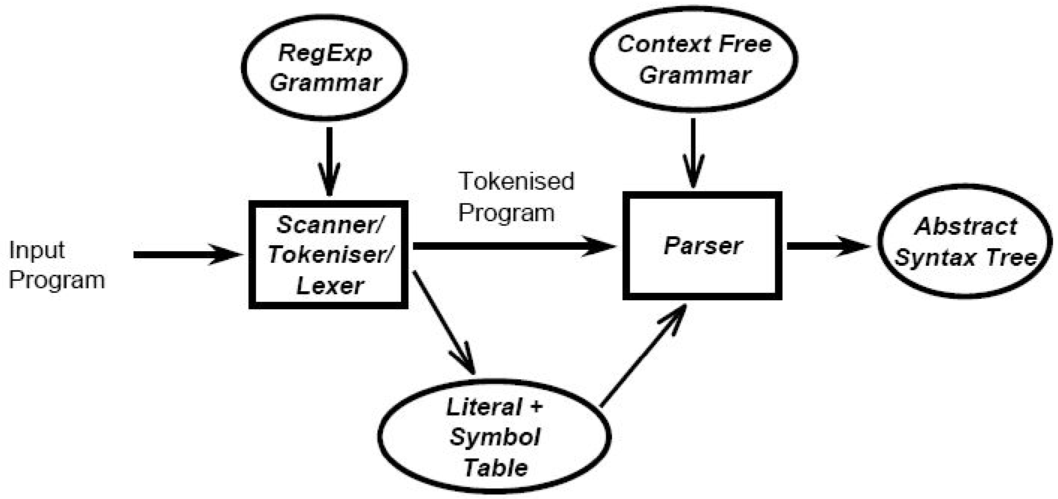
```
<program>
	<field_declarations count=“2”>
		Normal
		<declaration name=”field” type=”boolean”/>
		Array
		<declaration name=”x” count=“10” type=”integer”/>
	</field_declarations>
	<methods count=”1”>
		<method name=”foo” return_type=”int”>
			<params>
				<param name=”y” type=”int”>
			</params>
			<locals>
			</locals>
			<body>
			</body>
		</method>
	</methods>
</program>
```

Steps:

1. Create a base class "Node".
2. Create a class corresponding to every non-terminal symbol in grammar. All these classes will inherit Node class. Every class contains a Node* for each of its children.
3. In bison file create a node corresponding to every production rule and assign it to $$ so that it can be passed to its parent rule. E.g. :

```
Identifier1
        :       Identifier
                { $$ = new Identifier1Node(new IdentifierNode($1), NULL); }
        |       Identifier OpenSquare Int_literal CloseSquare
                { $$ = new Identifier1Node(new IdentifierNode($1), $3); };
```

4. Now follow visitor pattern to display output corresponding to every node. It'll help to avoid type castings to be done manually from Node* to any of its child classes. Type casting is required so that we can map correct display() method to each node. The task is can be done easily using visitor patterns. It utilizes run time binding using virtual functions and function overloading technique to avoid explicit type castings.
5. Implement the functionality to display appropriate output in corresponding visit methods.



Issues faced:

1. Deciding about class structure for every node was a challenging task. But solution was simple to implement if we just create a class for every symbol on left of production and making the symbols on right as its members /children.
2. Visiting pattern made the task of calling appropriate visit method depending on node value, simple but to generate the correct XML structure some effort was required. As

we want to display in such a way that output represent the program in easily understandable form.

References:

1. http://ashimg.tripod.com/Parser.html
2. http://aquamentus.com/flex_bison.html
3. https://en.wikipedia.org/wiki/Visitor_pattern

Code repository

https://github.com/thegame61916/DecafCompilerProject