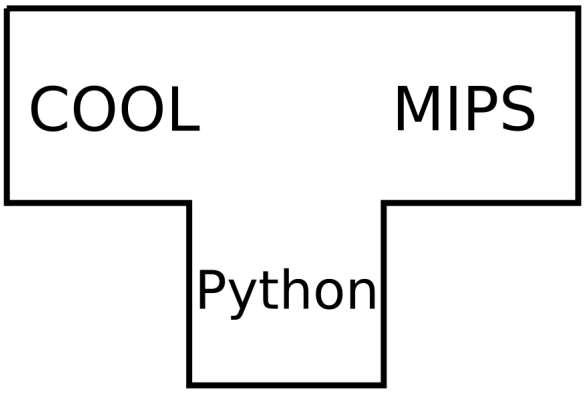
CS335 Assignment 0

Abhishek Kumar Madhukant Tushar Gupta
150035 150368 150771
abhikr@iitk.ac.in mkant@iitk.ac.in tusharg@iitk.ac.in

January 21, 2018

1. **T Diagram**



2. **BNF**

$\langle program \rangle ::= \langle imports \rangle \langle classes \rangle$

$\langle imports \rangle ::= import\ ID;\ \langle imports \rangle | \langle empty \rangle$
$\langle classes \rangle ::= \langle classes \rangle \langle class \rangle;$
$\quad | \langle class \rangle;$

$\langle class \rangle ::= class\ TYPE \langle inheritance \rangle \{ \langle features\_list\_opt \rangle \};$

$\langle inheritance \rangle ::= inherits\ TYPE \mid \langle empty \rangle$

$\langle features\_list\_opt \rangle ::= \langle features\_list \rangle | \langle empty \rangle$

$\langle features\_list \rangle ::= \langle features\_list \rangle \langle feature \rangle;$
$\quad | \langle feature \rangle;$

$\langle feature \rangle ::= \langle modifier \rangle\ ID(\langle formal\_params\_list\_opt \rangle) : TYPE \{ \langle expression \rangle \}$
$\quad | \langle modifier \rangle \langle formal \rangle$
$\langle modifier \rangle ::= \ public \mid private \mid \langle empty \rangle$

$\langle formal\_params\_list\_opt \rangle ::= \langle formal\_params\_list \rangle$
$\quad | \langle empty \rangle$

$\langle formal\_params\_list \rangle ::= \langle formal\_params\_list \rangle, \langle formal\_param \rangle$
$\quad | \langle formal\_param \rangle$

$\langle formal\_param \rangle ::= ID : TYPE$
$\quad | ID[\ ] : TYPE$
$\langle formal \rangle ::= ID : TYPE <\!\!- \langle expression \rangle ;$
$\quad | ID : TYPE ;$
$\quad | ID : TYPE\ [\langle experssion \rangle] ;$
$\langle expression \rangle ::= ID <\!\!- \langle expression \rangle$
$\quad | ID[\langle expression \rangle] <\!\!- \langle expression \rangle$
$\quad | \langle expression \rangle . ID(\langle arguments\_list\_opt \rangle)$
$\quad | \langle expression \rangle @ TYPE . ID(\langle arguments\_list\_opt \rangle)$
$\quad | \langle if\_then\_else \rangle$
$\quad | \langle while \rangle$
$\quad | \langle for \rangle$
$\quad | \langle block\_expression \rangle$

$$
\begin{aligned}
&|\langle let\_expression\rangle \\
&|newTYPE \\
&|isvoid\langle expr\rangle \\
&|\langle expression\rangle + \langle expression\rangle \\
&|\langle expression\rangle - \langle expression\rangle \\
&|\langle expression\rangle * \langle expression\rangle \\
&|\langle expression\rangle / \langle expression\rangle \\
&|\langle expression\rangle \% \langle expression\rangle \\
&| \langle expression\rangle \\
&|\langle expression\rangle < \langle expression\rangle \\
&|\langle expression\rangle <= \langle expression\rangle \\
&|\langle expression\rangle = \langle expression\rangle \\
&|\langle expression\rangle > \langle expression\rangle \\
&|\langle expression\rangle >= \langle expression\rangle \\
&|\langle expression\rangle or \langle expression\rangle \\
&|\langle expression\rangle and \langle expression\rangle \\
&|not\langle expression\rangle \\
&|(\langle expression\rangle) \\
&|SELF \\
&|ID \\
&|ID[\langle expression\rangle] \\
&|INTEGER \\
&|STRING \\
&|TRUE \\
&|FALSE \\
&|BREAK \\
&|CONTINUE
\end{aligned}
$$

$$
\begin{aligned}
\langle arguments\_list\_opt\rangle ::= &\langle arguments\_list\rangle \\
&|\langle empty\rangle
\end{aligned}
$$

$$
\begin{aligned}
\langle arguments\_list\rangle ::= &\langle arguments\_list\_opt\rangle, \langle expression\rangle \\
&|\langle expression\rangle
\end{aligned}
$$

$$
\langle action\rangle ::= ID : TYPE => \langle expr\rangle
$$

$$
\begin{aligned}
\langle actions\rangle ::= &\langle action\rangle \\
&|\langle action\rangle\langle actions\rangle
\end{aligned}
$$

$$
\langle if\_then\_else\rangle ::= if\langle expression\rangle then\langle expression\rangle else\langle expression\rangle fi
$$

$\langle while \rangle ::= while \langle expression \rangle loop \langle expression \rangle pool$
$\langle for \rangle ::= for(\langle expression \rangle; \langle expression \rangle; \langle expression \rangle) loop \langle expression \rangle pool$

$\langle block\_expression \rangle ::= \langle block\_list \rangle$

$\langle block\_list \rangle ::= \langle block\_list \rangle \langle expression \rangle;$
$\quad | \langle expression \rangle;$

$\langle let\_expression \rangle ::= let \langle formal \rangle in \langle expression \rangle$
$\quad | \langle nested\_lets \rangle, \langle formal \rangle$

$\langle nested\_lets \rangle ::= \langle formal \rangle IN \langle expression \rangle$
$\quad | \langle nested\_lets \rangle, \langle formal \rangle$

$\langle empty \rangle ::=$

3. **Syntactic rules deleted from base Language**
   $\langle case \rangle ::= case \langle expression \rangle of \langle actions \rangle esac$

4. **Reserved Keyword**

   - Int
   - String
   - Object
   - and
   - case
   - class
   - else
   - esac
   - fi
   - if
   - import
   - in
   - inherits
   - isvoid
   - for

- let
- loop
- new
- not
- or
- of
- pool
- public
- private
- self
- then
- while

5. **Semantic description of the new constructs added**

   - **import**
     import statement is used to bring certain classes or the entire packages, into visibility. For example if we have to create a list of integers, we can import a list package(already written COOL file) and create a list object and use functions on it

     ```
     import List;

     ....

     L = new List;
     L.append(1);
     ```

     This code compiles correctly if we have package named List.cl and has append function defined in it. *import* declaration section is a compile-time element of the source codes, and has no presence at run-time

   - **for-loop**
     A *for* loop, as described in the BNF above, has the following structure:

     ```
     for(expr_1; expr_2; expr_3)
     loop
         expr_block;
     pool
     ```

     The execution protocol is as follows:

- expr_1 is initialization, executed only once before the execution of the for loop
- expr_2 is condition, evaluated before each iteration of the loop. If expr_2 evaluates to **TRUE**, then the expr_block is executed. However, if expr_2 is evaluated to be **FALSE**, for loop is broken out of and control reaches the statement after the *pool* keyword
- expr_3 is updation, executed after the execution of each iteration of the loop

For example, the code below:

```
for(i<-0; i<5; i<-i+1)
loop{
    out_int(i)
}pool
```

outputs:

```
01234
```

- **Access Modifiers** We allow two access modifiers - *public, private.* By default all the classes, methods and variables in class are public. Class variables can be made private barring direct access to these variables from outside the class(will need getter and setter functions). A method can be declared as private can be called from inside the class only.
- **Array** An array is a group of like-typed variables that are referred to by a common name. This language only provide static allocation of array. Array can be declared by -

```
arr : Int[10];
```

Its assignment is done by

```
arr[0] <- 5;
arr[1] <- 6;
.
.
```

- **BREAK & CONTINUE**

BREAK is used to exit from the current running loop.
When the break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

CONTINUE is used to skip the current iteration in loop.
In for loop, after the continue keyword control reaches to the updation step of iterating variable In while loop after the continue keyword control flow reaches to the boolean check expression

- **Operator** Some operator are added to the original COOL language. Operators added are -
  - % - It is the modulus operator which gives the remainder of the two operands-

    ```
    c <- a % b
    ```

    variable c gets remainder of division when variable a is divided by variable b.
  - $>=, >$ these are the general "greater than or equal to" and "greater than" operators
  - **or** is used as the general 'logical or' operator on two boolean expressions.
  - **and** is used as the general 'logical and' operator on two boolean expressions.

6. **Tool** to be used for lexing and parsing are from PLY library of Python. We may use some other Python libraries if needed.