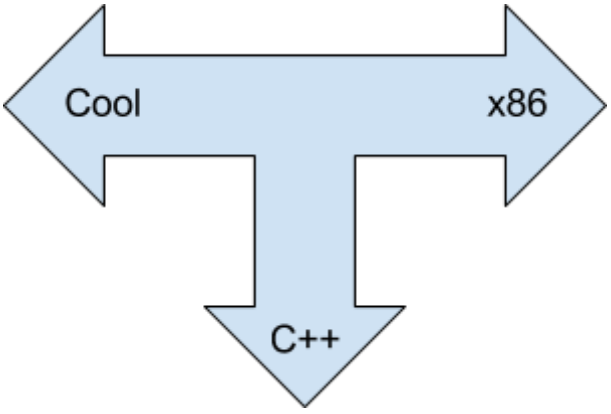
Assignment 0 (CS335A)

1. Group Details

Name	Roll No.	email-ID
Deepanshu Bansal	150219	dbansal@iitk.ac.in
Anupriy	150121	anupriy@iitk.ac.in
Jaskirat Singh	150302	jaskis@iitk.ac.in

2. T-diagram



3. EBNF of our language

[[…]]+ : one or more [[…]] : zero or more […] : optional*

LEXICAL RULES :

<Letter>	<Capital_letter> | <Small_letter>
<Capital_letter>	A | B | ... | Z
<Small_letter>	a | b | ... | z
<Digit>	0 | 1 | ... | 9
<ID>	<Small_letter> [[<Letter> | Digit | _]]*
<TYPE>	<Capital_letter> [[<Letter> | Digit | _]]*
<INTEGER>	[[<Digit>]]+
<STRING>	[[<ch>]]+ where ch is any printable ASCII character other than double quotes and the newline character

SYNTAX RULES :

<Compilation_unit>	[<Package_declaration>] [<Import_declarations>] [<Program>]
<Package_declaration>	package <Package_name> ;
<Package_name>	<Package_name>.<ID> | <ID>
<Import_declarations>	<Import_declarations> <Import_declaration> | <Import_declaration>
<Import_declaration>	import <Package_name>; | import <Package_name>.*;
<Program>	[[<Class>; | <Interface>;]]*
<Class>	class <TYPE> <Inheritance> <Implement_Interface> { <Features_list_opt> }
<Interface>	interface <TYPE> <Interface_Inheritance_List> { <Interface_features_list_opt> }
<Interface_Inheritance_List>	inherits <TYPE> | <Interface_Inheritance_List>, <TYPE> | <Empty>
<Inheritance>	inherits <TYPE> | <Empty>
<Implement_Interface>	implements <Interfaces_list> | <Empty>
<Interfaces_list>	<Interfaces_list>, <TYPE> | <TYPE>
<Features_list_opt>	<Features_list> | <Empty>
<Features_list>	<Features_list> <Feature> ; | <Feature> ;
<Feature>	<ID> (<Formal_params_list_opt>) : <TYPE> { <Expression> } | <Formal>
<Interface_features_list_opt>	<Interface_features_list> | <Empty>
<Interface_features_list>	<Interface_features_list> <Interface_feature> ; | <Interface_feature> ;
<Interface_feature>	<ID> (<Formal_params_list_opt>) : <TYPE>
<Formal_params_list_opt>	<Formal_params_list> | <Empty>
<Formal_params_list>	<Formal_params_list> , <Formal_param> | <Formal_param>
<Formal_param>	<ID> : <TYPE> ['['']']
<Formal>	<ID> : <TYPE> ['[' [<Expression>] ']'] <- <Expression> | <ID> : <TYPE> ['[' <Expression> ']']
<Expression>	<ID> <- <Expression> | <Expression>.<ID>(<Arguments_list_opt>) | <Expression>@<TYPE>.<ID>(<Arguments_list_opt>) | <Conditionals> | <Loops> | <Block_Expression>

	| <Let_Expression> | new <TYPE> | isvoid <Expression> | <Return_statement> | <Break_statement> | <Continue_statement> | <Expression> + <Expression> | <Expression> - <Expression> | <Expression> * <Expression> | <Expression> / <Expression> | ~ <Expression> | <Expression> < <Expression> | <Expression> <= <Expression> | <Expression> = <Expression> | <Expression> > <Expression> | <Expression> >= <Expression> | <Expression> && <Expression> | <Expression> || <Expression> | <Expression> & <Expression> | <Expression> | <Expression> | <Expression> ^ <Expression> | not <Expression> | (<Expression>) | SELF | <ID> | <ID> '[' <Expression> ']' | '[' [[<Expression>]]+ ']' | true | false | SELF_TYPE | <INTEGER> | <STRING>
<Conditionals>	<Case> | <If_then_else>
<Loops>	<While> | <For> | <Do_while>
<Arguments_list_opt>	<Arguments_list> | <Empty>
<Arguments_list>	<Arguments_list> , <Expression> | <Expression>
<Case>	case <Expression> of <Actions> esac
<Actions>	<Action> | <Action> <Actions>
<Action>	<ID> : <TYPE> => <Expression>
<If_then_else>	if <Expression> then <Expression> else <Expression> fi
<While>	while <Expression> loop <Expression> pool
<For>	for (<Expression> ; <Expression> ; <Expression>) loop <Expression> pool
<Do_while>	do loop <Expression> pool while <Expression>
<Break_statement>	break

<Continue_statement>	continue
<Return_statement>	return
<Block_Expression>	{ <Block_list> }
<Block_list>	<Block_list> <Expression> ; | <Expression> ;
<Let_Expression>	let <Formal>[[,<Formal>]]* in <Expression>
<Empty>	

COMMENTS : There are two forms of comments in Cool. Any characters between two dashes "--" and the next newline (or EOF, if there is no next newline) are treated as comments. Comments may also be written by enclosing text in (∗ . . . ∗). The latter form of comment may be nested. Comments cannot cross file boundaries.

4. Rules deleted

We have not deleted any rules from this language.

5. Semantic description of the new constructs added

a. Import Statements :
   - This is used to import the code from multiple files,especially libraries, similar to the java import function.
   - Syntax: import <Package name> ;

b. Package declaration :
   - This is a used to name a program. Ex: We write a program A. While writing another program B, it is possible we need to import the code of A into it. So, for that, we only need to call import <package name> which is specified during the package declaration of A.
   - Syntax: package <Package name>

c. Interfaces :
   - We can now add interfaces to our programs as well
   - Ex.

```
interface Vehicle {
        noOfWheels() : Int;
};
```

Now any class can implement this interface.
Ex.

```
class Scooter implements Vehicle {
        noOfWheels() : Int { 2 };
};
class Car implements Vehicle {
        noOfWheels() : Int { 4 };
};
```

d. Support for Arrays :
   - We can create arrays as follows :
     x : Int[3] <- [10,20,30]
     Now can access ith element just by x[i], ex. x[0] <- 10
   - Passing arrays as function(features) parameters is as follows :
     func (a : Int [ ]) : Int { a[0] + a[1] }

e. Break statement :
   - In a loop, to exit from the iterations of the loop.
   - while <condition> loop
     if <expr> then
     break
     else
     <Expression>
     fi
     pool

f. Continue statement :
   - In a loop,to skip the current iteration
   - while <condition> loop
     if <expr> then
     continue
     else
     <Expression>
     fi
     pool

g. Return statement :
   - By default, last statement of any scope (eg. features, block etc.) is returned
   - Using this one can return an expression which is before last expression in any feature(functions) like in C.
   - In the following example, if a equals 0, then 1 is returned from the function func, otherwise a is incremented and then a is returned as in the last expression.
   - func (a : Int) : Int {
     {
     if (a = 0) then
     return 1
     else
     a <- a + 1
     fi ;
     a ;
     }
     }

h. For loop :
   - This is like “for loop” as defined in C.

- <Expression> between loop and pool is repeated in this.
- First <Expression> in round braces is used for initialization purposes.
- Second <Expression> in round braces is used for check condition before each iteration.
- Third <Expression> in round braces is used for updation of variables after each iteration.
- for (i : Int <- 0 ; i < 335 ; i <- i+1) loop
  <Expression>
  pool

i. Do while loop :
   - Just another method for loop
   - Functionality similar to that in C.
   - x <- 10;
     do loop
     x <- x+1;
     pool while (x < 335)
     This will update value of x until it reach 335.

j. New Relational Operators :
   - <Expression> > <Expression>
   - <Expression> >= <Expression>
   - They serve as normal comparators.
   - These operators can only be applied to expressions which are of type Int.
   - Return type is Bool.
   - For ex : 2 >= 3 , returns Bool type, in this case which is false

k. Boolean operations :
   - <Expression> && <Expression>
   - Only true when both expressions are true,
   - <Expression> || <Expression>
   - True when at least one expression is true.
   - Type of both expressions must be Bool for boolean operators.
   - Return type is Bool

l. Some bitwise functions :
   - <Expression> & <Expression> (Bitwise AND)
   - <Expression> | <Expression> (Bitwise OR)
   - <Expression> ^ <Expression> (Bitwise XOR)
   - <Expression> must be of type Int and return type is Int

6. Tools we will be using in this project

a. Lexer Generator : flex

b. Parser Generator : bison