ACE DOCUMENTATION

NAME: **ACE**, file extension: ace

DESIGN:

Language type: Imperative

Data Types: ACE supports int, bool and string data types. You don't need to put data types while declaring the variable. The compiler will dynamically recognize the data type.
ex: var x = 0, var y = "team23"

Conditional Statements: ACE supports both traditional if and else statement and ternary operator.
ex:
```
if ( x>0 ){
    x = x+1
   }
-> var x = (x>0) ? { 1 : 0 }
```

Loops: ACE supports **while**, traditional **for loop** and **for loop with range.**
ex:
```
while (x>0){
    x = x +2
    }
    for(var x = 0; x<10;x++){
    out(x)
    }
    For i in range(5){
    out(i)
    }
```

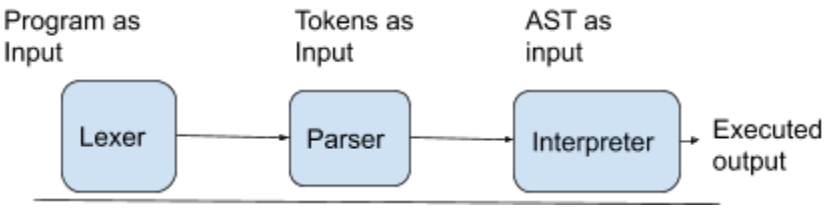
Operators: ACE supports the arithmetic operators(+,-,*,/), supports relational operators (>,<,<=,>=,==,!=), logical operators (or, and, not) and unary operators (++, --).

Print statements: The print statements in ACE is “out”.
ex: out(“team23”), out(x+y).

Functions: ACE supports function declaration at the top of the program.
ex:
```
fun add(x,y){
    send x+y
}
var x = add(3,2);
```

Parser: docstring is used, somewhat unusually, to bind grammar rules to functions. The effects of reductions can be described very conveniently in the function body. An Abstract Syntax Tree will be produced using Yacc as a tuple. **YACC** uses LALR(**Look Ahead Left-Right**) parsing technique which is a **bottom-up approach**.

Interpreter: The Environment changes are maintained using a Dictionary to lookup and update the variables. Another dictionary is used to maintain to lookup for functions declared by the user.



GRAMMAR:

1)P ::= FUNCTIONLIST K | K

2) FUNCTIONLIST ::= F FUNCTIONLIST | empty
3)F::= fun IDENT (Arg) { K send E} | fun IDENT (Arg) { K send B } |
fun IDENT ( ) { K send }

4)Arg::= IDENT , Arg
5)Arg::= IDENT | empty

6)K ::= Statements K | statements

7)Statements ::= DECLARATION ;| INITIALIZATION ;| ASSIGN ;| IF | while B { K } |
FOR | Print ; | UNARY | FUNCALL;

8)DECLARATION::= var IDENT

9)INITIALIZATION::= var IDENT = E | var IDENT = S | var IDENT = FUNCALL

10)ASSIGN::= IDENT = E |IDENT= B | IDENT = S | IDENT = FUNCALL

11)UNARY ::= IDENT ++ | IDENT --

12) FUNCALL::= IDENT ( )

13)IF::= if B { K } ELIF | if E { K } ELIF

14)ELIF::= else if B { K } ELIF | else if E { K } ELIF

15)ELIF::=else { K }

16)ELIF::= empty

17)FOR::= for (INITIALIZATION ; B ; ASSIGN) { K } |
for (INITIALIZATION ; B ; UNARY) { K } |

for IDENT in range (E , E) { K }

18)Print::= out (“S”) | out (IDENT) | out (“S” , IDENT).

19)B ::= true | false | not B | CONDITION | B or B | B and B | E

20)C ::= E < E | E > E | E<= E | E >= E | E == E

21)E ::= E + E | E - E | E * E | E / E | IDENT | N | TERNARY

22)IDENT ::= [A-Z] IDENT1

23)IDENT1 ::= [A-Z] | [0 - 9] | empty

24)N ::= D N

25)N ::= D

26)D ::= [0-9]

27) TERNARY::= (B) ? (E : E)

GITHUB REPOSITORY:

https://github.com/RohithVarmaGaddam/SER502-Spring2020-Team23

References:

1) https://www.matthieuamiguet.ch/media/documents/TeachingCompilersWithPython_Paper.pdf

2) http://www.dabeaz.com/ply/ply.html#ply_nn4