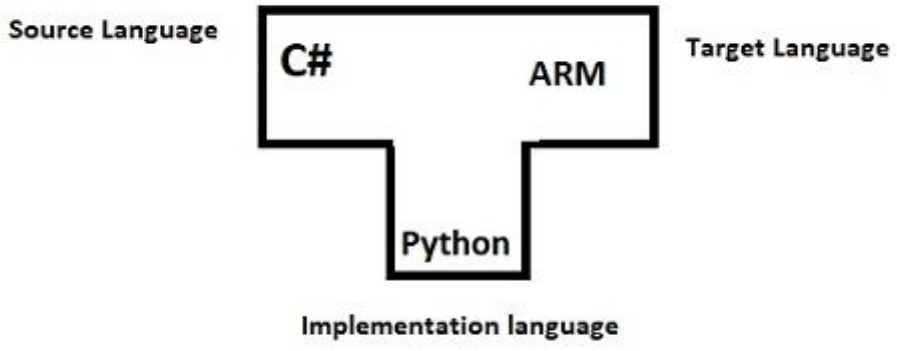
CS335 Assignment-0

Group Members:

Nikhil Ghantudiya(14246), nikhildg@iitk.ac.in
Kamlesh Kumar Meena(14299), meena@iitk.ac.in
Ankit Gupta(14103), ankgupta@iitk.ac.in

T-Diagram



BNF:

```
{

  tokens=[
     LBRACE             =  '{'
     RBRACE             =  '}'
     LBRACK             =  '['
     RBRACK             =  ']'
     LPAREN             =  '('
     RPAREN             =  ')'
     COLON              =  ':'
     SEMI               =  ';'
     COMMA              =  ','
     EQ                 =  '=='
     ASSIGN             =  '='
     NOT_EQ             =  '!='
     NOT                =  '!'
     PLUS_PLUS          =  '++'
     PLUS_ASSIGN        =  '+='
     PLUS               =  '+'
     MINUS_MINUS        =  '--'
     MINUS_ASSIGN       =  '-='
     MINUS              =  '-'
     COND_OR            =  '||'
     BIT_OR_ASSIGN      =  '|='
     BIT_CLEAR_ASSIGN   =  '&^='
     BIT_CLEAR          =  '&^'
     COND_AND           =  '&&'
     BIT_AND_ASSIGN     =  '&='
     BIT_AND            =  '&'
     BIT_OR             =  '|'
     SHIFT_LEFT_ASSIGN  =  '<<='
     SHIFT_LEFT         =  '<<'
     LESS_OR_EQUAL      =  '<='
     LESS               =  '<'
     BIT_XOR_ASSIGN     =  '^='
     BIT_XOR            =  '^'
     MUL_ASSIGN         =  '*='
     MUL                =  '*'
     QUOTIENT_ASSIGN    =  '/='
     QUOTIENT           =  '/'
     REMAINDER_ASSIGN   =  '%='
     REMAINDER          =  '%'
     SHIFT_RIGHT_ASSIGN =  '>>='
     SHIFT_RIGHT        =  '>>'
     GREATER_OR_EQUAL   =  '>='
```

```
    GREATER          =  '>'
    DOT              =  '.'
    APOS             =  "'"
    QUOTE            =  '"'
    BACK             =  '\'
    MOD              =  '%'

    WS               = 'regexp:\s+'
    COMMENT          = 'regexp://.*'
    DIGIT            = 'regexp:\d+(\.\d*)?'
    STRING           = "regexp:('([^'\\]|\\.)*'|\"([^\"\\]|\\.)*\")"
    ID               = 'regexp:\p{Alpha}\w*'
  ]
}
```

NamespaceDec ::= namespace ID LBRACE ClassList RBRACE
ClassList ::= ClassDec*
ClassDec ::= (AccessModifier? TypeModifier? Modules ID LBRACE MethodList RBRACE)
MethodList ::= (MethodDec | EnumDec)*
MethodDec ::= AccessModifier? TypeModifier* (Type | void) ID LPAREN (Type ID ','?)* RPAREN LBRACE Expr* RBRACE
EnumDec ::= AccessModifier? enum ID LBRACE (ID ',')* ID* RBRACE

Expr ::= (Arithmetic | AssignArithmetic | Using | Return | ForStatement | ForEachStatement | IfStatement | FlowStatement | MethodCall | TypeCheck)

GeneralDec ::= (PrimitiveTypes | GenericTypes)? (CustomType+) ASSIGN Cast? (Arithmetic | Invocation | TypeCheck | MethodCall | Bool | null | DIGIT | CustomType) SEMI?

NewTypeDec ::= ((PrimitiveTypes | CustomType) ID | Property) ASSIGN new?(PrimitiveTypes | GenericTypes | MethodCall) SEMI?

BoolDec ::= bool ID ASSIGN (true | false) SEMI {pin=2}
CharDec ::= char ID ASSIGN APOS ([a-zA-Z0-9] | (BACK ('u' | 'x') DIGIT{4})) APOS SEMI {pin=2}
IntDec ::= u?(int | long | short) ID ASSIGN (Arithmetic | DIGIT) SEMI {pin=3}
StringDec ::= 'string' ID ASSIGN STRING SEMI {pin=2}

Arithmetic ::= Cast? (MethodCall | Property| DIGIT | CustomType) (Operator Cast? (MethodCall | Property| DIGIT | CustomType))+
AssignArithmetic ::= CustomType AssignOperator (MethodCall | Property | CustomType | DIGIT | STRING)? SEMI?
Comparison ::= (Expr | CustomType | DIGIT | STRING) (CompareOperator (Expr | CustomType | DIGIT | STRING))+

```
Parameter ::= ID LPAREN ((PrimitiveTypes | CustomType) ID ','*)* RPAREN
Argument ::= LPAREN (out? Cast? (Invocation | Arithmetic| MethodCall | Property | DIGIT |
STRING | null | Bool | TypeCheck | CustomType) ','?)* RPAREN
Invocation ::= new (PrimitiveTypes | GenericTypes | CustomType) Argument?

MethodCall ::= (ID) (('.'(CustomType))? Argument)+ SEMI?
Property ::= (MethodCall | CustomType) ('.' ID)+ Array? SEMI?

ForStatement ::= for LPAREN IntDec Arithmetic SEMI? AssignArithmetic RPAREN
LBRACE? Expr* RBRACE? {pin=10}
ForEachStatement ::= foreach LPAREN (PrimitiveTypes | CustomType) ID in (Property |
MethodCall | CustomType) RPAREN LBRACE? Expr* RBRACE? {pin=8}
Using ::= using LPAREN CustomType ID ASSIGN (Invocation | MethodCall | Property)
RPAREN LBRACE Expr* RBRACE {pin=9}
Return ::= return (Expr | Bool | CustomType) SEMI?
IfStatement ::= if LPAREN (Comparison | TypeCheck | CustomType) RPAREN LBRACE?
Expr* RBRACE? ElseIfStatement* ElseStatement?
private ElseStatement ::= else LBRACE? Expr* RBRACE?
private ElseIfStatement ::= else IfStatement

FlowStatement ::= (break | continue) SEMI

TypeCheck ::= ID () (PrimitiveTypes | GenericTypes | CustomType) // todo name + // remove
from expr?

private Cast ::= LPAREN (PrimitiveTypes | GenericTypes | CustomType) RPAREN

private Array ::= LBRACK (Arithmetic | ID | DIGIT | STRING)? RBRACK

Type ::= PrimitiveTypes | GenericTypes | CustomType
Value ::= Bool | STRING | DIGIT | null



/* Tokens */ // temp
PrimitiveTypes ::= (bool | char | int |  object | 'string') Array?    // todo

Bool ::= true | false
ValueClasses ::= (struct | enum)
Modules ::= (class )

PreprocessorDec ::= '#'(if | else | elif )

AccessModifier ::= (public)
TypeModifier ::= (const | new ) // todo split into method modifiers w/ rules
```

ReservedWords ::= (break | continue | else | false | for | foreach | if | namespace | new | return | true | typeof | while)

Operator ::= PLUS | MINUS | QUOTIENT | BIT_AND | BIT_OR | BIT_XOR | GREATER | GREATER_OR_EQUAL | LESS | LESS_OR_EQUAL | MOD
AssignOperator ::= PLUS_ASSIGN | MINUS_ASSIGN | MUL_ASSIGN | QUOTIENT_ASSIGN | REMAINDER_ASSIGN | BIT_OR_ASSIGN | BIT_AND_ASSIGN | BIT_XOR_ASSIGN

CompareOperator ::= EQ | NOT_EQ | LESS | LESS_OR_EQUAL | GREATER | GREATER_OR_EQUAL | COND_OR | COND_AND

Syntactic Rules Deleted:
File ::= ImportList NamespaceDec
ImportList ::= ImportDec*
ImportDec ::= using ID (ID | DOT)* SEMI
Expr ::= (VarDec | QualifiedClassDec | Switch | Property | Exception)

VarDec ::= AccessModifier? TypeModifier? (BoolDec | ByteDec | CharDec | FloatDec | IntDec | StringDec | NewTypeDec | ListDec | GeneralDec | PreprocessorDec | CustomObjectDec | DictionaryDec)

NewTypeDec ::= (( ) ASSIGN new? (QualifiedClassDec) SEMI?
ByteDec ::= byte ID ASSIGN (0x DIGIT+) SEMI {pin=2} // todo
FloatDec ::= float ID ASSIGN DIGIT '.'? 'f' SEMI {pin=2}
ListDec ::= GenericTypes ID ASSIGN ((new GenericTypes) | MethodCall) SEMI?
DictionaryDec ::= GenericTypes ID ASSIGN (((new GenericTypes | TypeCheck)) | MethodCall | CustomType) SEMI
QualifiedClassDec ::= 'System' ('.' (PrimitiveTypes | GenericTypes | ID))+ Argument SEMI {pin=2}
CustomObjectDec ::= CustomType ID ASSIGN (Cast (ID | DIGIT | STRING) | new CustomType LPAREN (ID | DIGIT | STRING)RPAREN) SEMI {pin=2}

Argument ::= LPAREN (out? Cast? (QualifiedClassDec) ','?)* RPAREN
Invocation ::= new (QualifiedClassDec) Argument?

MethodCall ::= (QualifiedClassDec) (('.'(QualifiedClassDec))? Argument)+ SEMI?
Switch ::= switch LPAREN (MethodCall | Property | CustomType) RPAREN LBRACE (case LPAREN? (DIGIT | STRING | CustomType) RPAREN? ':' Expr*)* (default ':' Expr*)? RBRACE

Exception ::= throw new MethodCall

TypeCheck ::= ID (is | as | instanceof) (QualifiedClassDec) // todo name + // remove from expr?
PrimitiveTypes ::= (byte | sbyte | decimal | double | float | uint | long | ulong | short | ushort | var) Array?   // todo
GenericTypes ::= (List | Dictionary) '<' (','? (PrimitiveTypes | GenericTypes | CustomType))+ '>' Argument?
private CustomType ::= (ID Array?) | null

Modules ::= (interface | delegate)

PreprocessorDec ::= '#'(endif | define | undef | warning | error | line | region | endregion | pragma | pragma warning | pragma checksum) ID*
AccessModifier ::= (private | protected | internal)

TypeModifier ::= (abstract | async | event | extern | override | partial | readonly | sealed | static | unsafe | virtual | volatile) // todo split into method modifiers w/ rules

ReservedWords ::= (as | base | case | catch | do | finally | goto | in | is | lock | null | out | ref | sizeof | stackalloc | switch | this | throw | true | try | typeof | using | var | yield)

AssignOperator ::= PLUS_PLUS | MINUS_MINUS

Tools:
Python PLY(Python Lex-Yacc) library

Features Not Included :

1. Simple library functions(sin,cos ,log, sqrt)
2. Switch case and do-while
3. Multidimensional Array
4. Array of Structures
5. Comments
6. Increment/decrement
7. File Import
8. Data types such as float,byte,list,dictionary
9. Exception Handler