ChucK Compiler

Team A
Zachary May
Jade Webb
Ajita Shrivastava
Yinuo (Melinda) Tang

Context

ChucK is a strongly-typed programming language similar to Java or C

It features easy-to-use functionalities for teaching the basic concepts of security.

A new randomly-named language which will have .ck extension.

Operators for Caesar Cipher, Vigenere Cipher, and Frequency Analysis which can be written just as easily as addition, subtraction.

Source Language

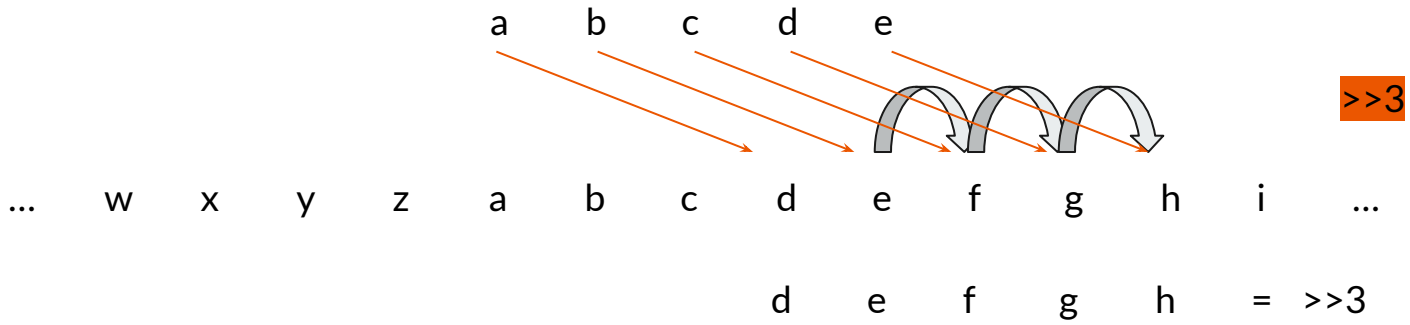
Devised a language that can be used for cyber security which supports:

- Primitive data types: Integer (I), String (S), Double (D)
- Operators: Multiplicative, arithmetic, relational, and operators for built-in functions
- Built-in functions: String Analysis (@), Encryption (<<, >>)
- Assignment Statement: using the (=) operator
- Conditional control statement: if-then-else statement
- Looping control statement: while loop
- Functions: pass by value and return

Built-in Function: Encryption

Operator ‘>>’ can be used to create a caesar-cipher effect on a string.

I.e. a shift of 3 on string “abcde” would convert it to “defgh” (>>3).



Built-in Function: Encryption

Operator ‘<<’ vigenere-cipher effect on a string. This functions similarly to the previous shift cipher, with the main difference being the use of a key string to determine the amount shifted for each letter.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	D
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	C
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	B
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Built-in Function: String Analysis

Operator '@' can be used with a string variable to calculate its character frequency with a single command.

```
S line;
line = 'Mississippi';
print[line @];
```

i	4
s	4
p	2
m	1

Grammar - Statements

Assignment statement

If statement

While statement

Function call

CK.g4 | sample.ck

```
grammar CK;

@header {
    package antlr4;
    import java.util.HashMap;
    import intermediate.symtab.SymtabEntry;
    import intermediate.type.Typespec;
}

program           : programHeader compoundStatement ;
programHeader     : PROGRAM programIdentifier '[' programParameters ']';
programParameters : IDENTIFIER ( ',' IDENTIFIER )* ;

programIdentifier   locals [ SymtabEntry entry = null ]
    : IDENTIFIER ;

statement : compoundStatement
          | assignmentStatement
          | ifStatement
          | whileStatement
          | printStatement
          | functionDefinitionStatement
          | functionCallStatement
          | variableDeclarationStatement
          | emptyStatement
          ;

variableDeclarationStatement : typeIdentifier variableIdentifier;
typeIdentifier : IDENTIFIER;
variableIdentifier locals [ Typespec type = null, SymtabEntry entry = null ] : IDENTIFIER;

compoundStatement : '{' statementList '}' ;
emptyStatement : ;

statementList       : statement ( ';' statement )* ;
assignmentStatement : lhs '=' rhs ;

lhs locals [ Typespec type = null ]
    : variable ;
rhs : expression ;

ifStatement    : IF '[' expression ']' trueStatement ( ELSE falseStatement )? ;
trueStatement  : statement ;
falseStatement : statement ;

whileStatement  : WHILE '[' expression ']' statement ;

functionCallStatement : functionCall ;

argumentList : argument ( ',' argument )* ;
argument     : expression ;

printStatement   : PRINT '[' expression ']' ;
```

Grammar - Expressions

Cypher operator - expression rule modified

String analysis - factor rule modified

Function definition (continued from previous slide)

CK.g4

```
expression          locals [ Typespec type = null ]
    : relationExpression (cypherOp relationExpression)? ;

relationExpression        locals [ Typespec type = null ]
    : simpleExpression (relOp simpleExpression)? ;

sign : '-' | '+' ;

simpleExpression    locals [ Typespec type = null ]
    : sign? term (addOp term)* ;

term                locals [ Typespec type = null ]
    : factor (mulOp factor)* ;

factor              locals [ Typespec type = null ]
    : variable              # variableFactor
    | number                # numberFactor
    | characterConstant     # characterFactor
    | stringConstant        # stringFactor
    | functionCall          # functionCallFactor
    | NOT factor            # notFactor
    | '[' expression ']'    # bracketedFactor
    | variable '@'          # stringAnalysis
    ;

variable            locals [ Typespec type = null, SymtabEntry entry = null ]
    : variableIdentifier ;

functionDefinitionStatement     locals [ Typespec type = null, SymtabEntry entry = null ]
    : FUNCTION typeIdentifier functionName '[' defArgumentList? ']' statement ;
defArgumentList : typeIdentifier variable (',' typeIdentifier variable)* ;
functionCall : functionName '[' argumentList? ']' ;
functionName        locals [ Typespec type = null, SymtabEntry entry = null ]
    : IDENTIFIER ;

number          : sign? unsignedNumber ;
unsignedNumber  : integerConstant | doubleConstant ;
integerConstant : INTEGER ;
doubleConstant  : DOUBLE;

characterConstant : CHARACTER ;
stringConstant    : STRING ;

relOp : '==' | '!=' | '<' | '<=' | '>' | '>=' ;
addOp : '+' | '-' | OR ;
mulOp : '*' | '/' | DIV | MOD | AND ;
cypherOp : '>>' | '<<' ;
```

Grammar - Tokens and Commenting

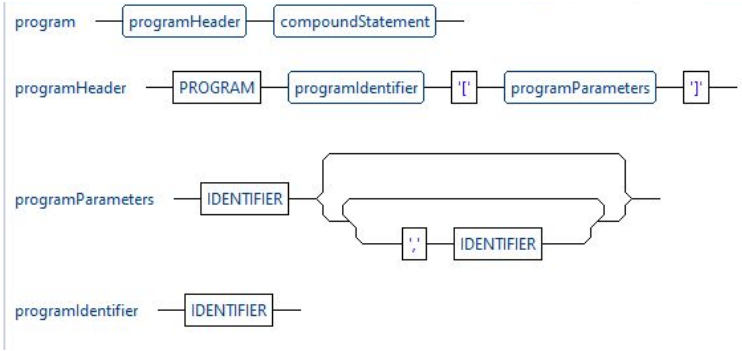
Comment - using ‘%’ to enclose comment statements.

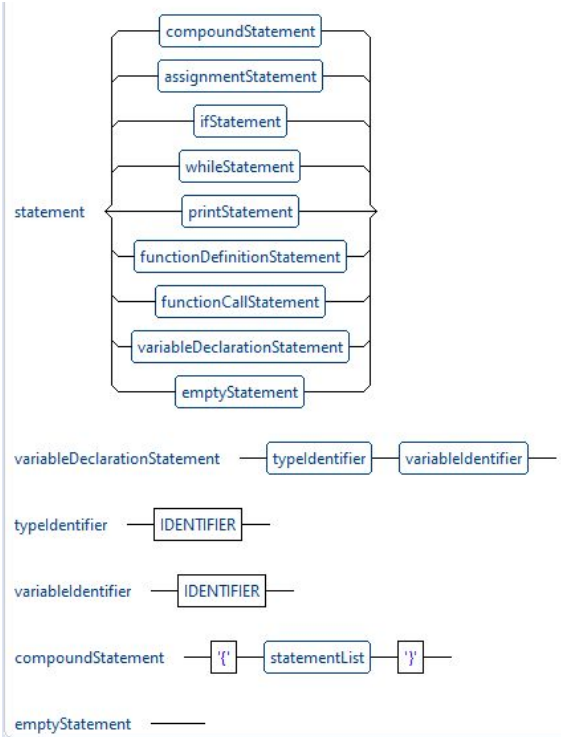
```
fragment A : ('a' | 'A') ;
fragment B : ('b' | 'B') ;
fragment C : ('c' | 'C') ;
fragment D : ('d' | 'D') ;
fragment E : ('e' | 'E') ;
fragment F : ('f' | 'F') ;
fragment G : ('g' | 'G') ;
fragment H : ('h' | 'H') ;
fragment I : ('i' | 'I') ;
fragment J : ('j' | 'J') ;
fragment K : ('k' | 'K') ;
fragment L : ('l' | 'L') ;
fragment M : ('m' | 'M') ;
fragment N : ('n' | 'N') ;
fragment O : ('o' | 'O') ;
fragment P : ('p' | 'P') ;
fragment Q : ('q' | 'Q') ;
fragment R : ('r' | 'R') ;
fragment S : ('s' | 'S') ;
fragment T : ('t' | 'T') ;
fragment U : ('u' | 'U') ;
fragment V : ('v' | 'V') ;
fragment W : ('w' | 'W') ;
fragment X : ('x' | 'X') ;
fragment Y : ('y' | 'Y') ;
fragment Z : ('z' | 'Z') ;

PROGRAM   : P ;
CONST     : C O N S T ;
DIV       : D I V ;
MOD       : M O D ;
AND       : A N D ;
OR        : O R ;
NOT       : N O T ;
IF        : I F ;
THEN      : T H E N ;
ELSE      : E L S E ;
WHILE     : W H I L E ;
PRINT     : P R I N T ;
FUNCTION  : F ;

IDENTIFIER : [a-zA-Z][a-zA-Z0-9]* ;
INTEGER    : [0-9]+ ;

DOUBLE     : INTEGER '.' INTEGER
           | INTEGER ('e' | 'E') ('+' | '-')? INTEGER
           | INTEGER '.' INTEGER ('e' | 'E') ('+' | '-')? INTEGER
           ;

NEWLINE : '\r'? '\n' -> skip  ;
WS      : [ \t]+ -> skip ;

QUOTE     : '\'' ;
CHARACTER : QUOTE CHARACTER_CHAR QUOTE ;
STRING    : QUOTE STRING_CHAR* QUOTE ;

fragment CHARACTER_CHAR : ~('\'')   // any non-quote character
                        ;

fragment STRING_CHAR : QUOTE QUOTE  // two consecutive quotes
                     | ~('\'')      // any non-quote character
                     ;

COMMENT : '%' COMMENT_CHARACTER* '%' -> skip ;

fragment COMMENT_CHARACTER : ~('%') ;
```

Syntax Diagrams

Program: P foo[n] { … }

Variable Declaration: S cypher;





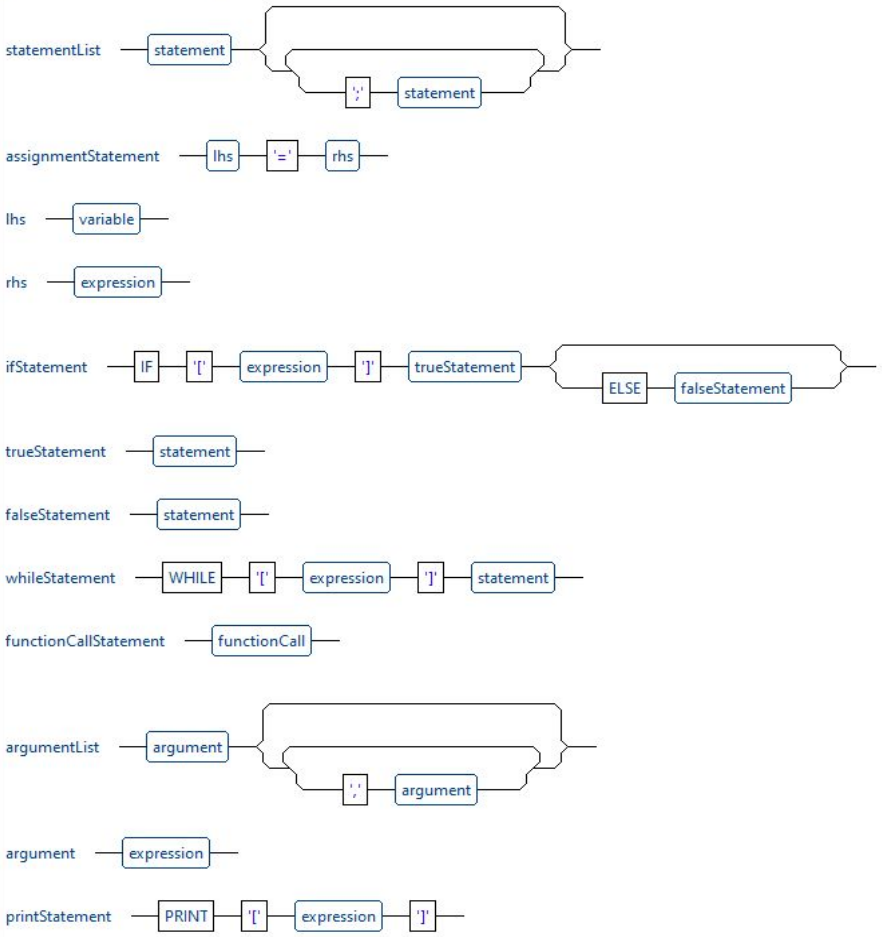
Syntax Diagrams

Assignment Statement: i = 4;

If Statement: if [i == 4] {i = i + 1} else {i = i - 1};

While Statement: while [n < 10] {n = n + 2};

Print Statement: print[n];



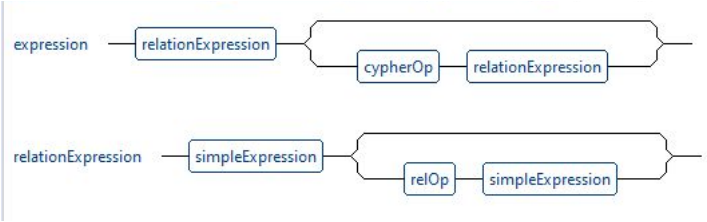
Syntax Diagrams

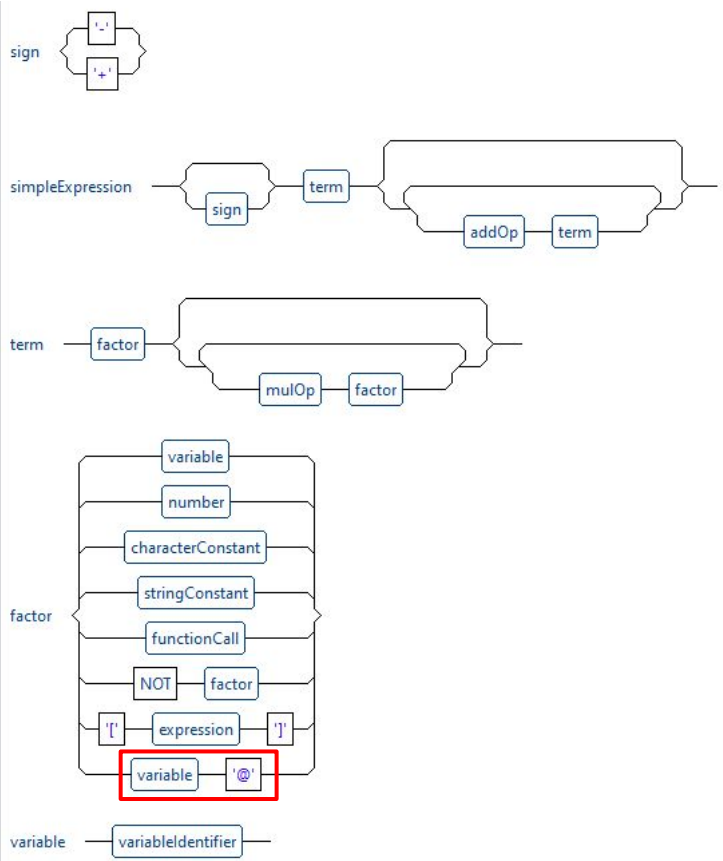
Expression: n >> 2

Relation Expression: i == 3

Simple Expression: i + 4

Term: i * i

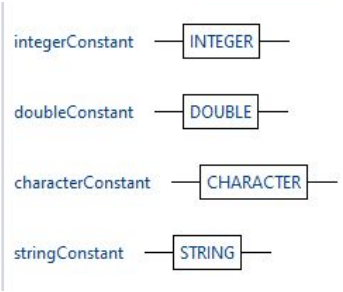


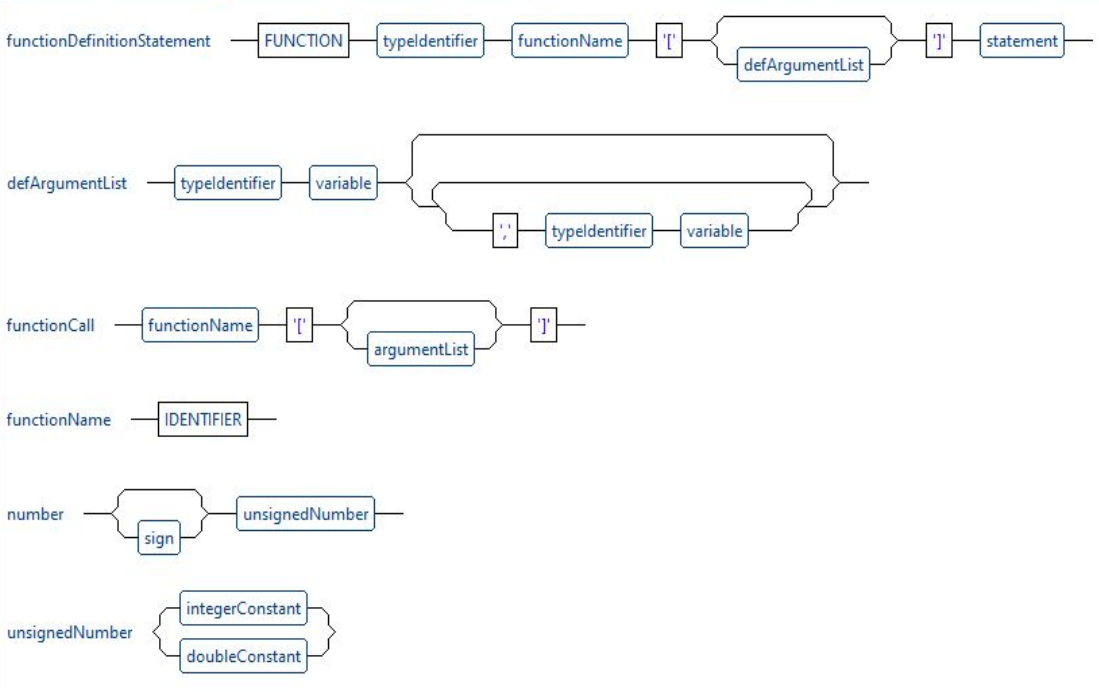


Syntax Diagrams

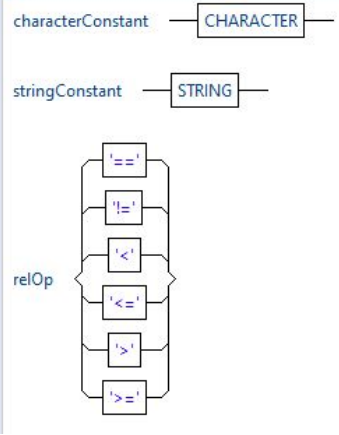
Function: F I foo[I count, S name] { … }

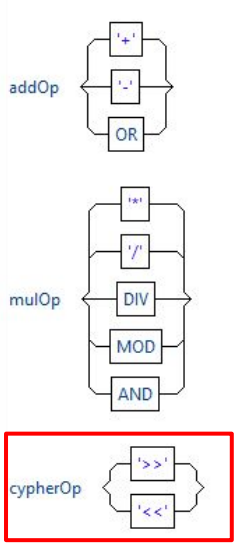
Function Call: foo[10, ‘test’];



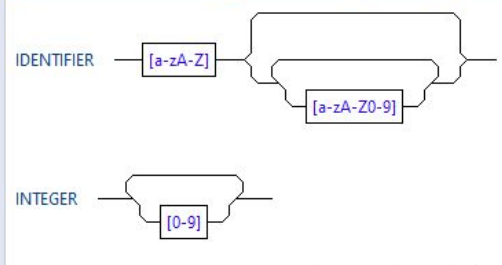


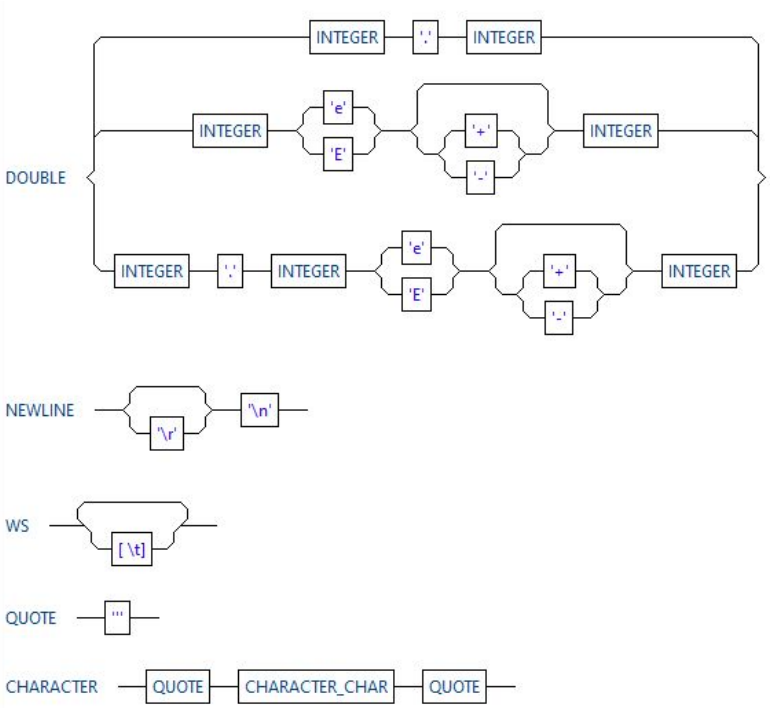
Syntax Diagrams





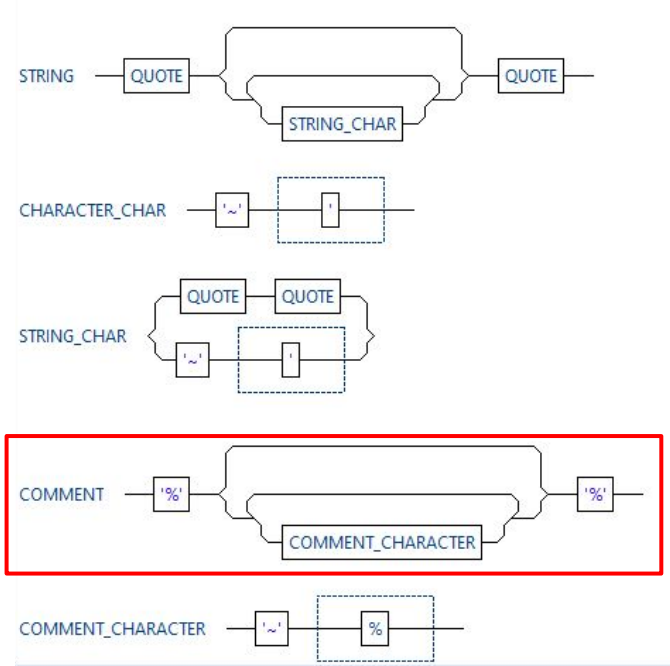
Syntax Diagrams





Syntax Diagrams

Comment: %this is a comment.%



Target Program - Features

Program definition

String definition

Print function

If - Then - Else statement

Function definition

Integer definition

While statement

```
P name[parameters]
{

S plain;
plain = 'The quick red fox jumped over the lazy brown dog';
S cypher;
cypher = plain >> 2; %encyphers the string by shifting by the amount given%
print[cypher];
S polyalphabet;
polyalphabet = plain << 'polyalphabet'; %encyphers the string based on the key given%

print[cypher @]; %String analysis%
print[polyalphabet @];

if[plain == deshift[2, cypher]]
{
        print['decyphered'];
}
else
{
        print['bad logic'];
};

I i;
i = 0;

while[i < 10]
{
        print[i];
        i = i + 1;
};

F S deshift[I amount, S cyphertext]
{
        deshift = cyphertext >> (26 - amount);
}

}
```

Simple Program

Program definition
String, Integer, Double
If Statement
Print Statement
Comments

TODO:
Variable accessing
Handling of <<, >>, @

```
P simple[parameters]
{

print['hello'];

S plain;
plain = 'The quick red fox jumped over the lazy brown dog';

I i;
i = 0;

D x;
x = 1;

if[i==0]{
    i = 2;
};

print['test'];

%print[i];%
%print[plain];%
}
```

Simple Program
- Jasmin Generation

Name	Date modified	Type	Size
bin	11/30/2022 11:19 PM	File folder	
Final	11/30/2022 1:33 PM	File folder	
target	11/30/2022 10:21 PM	File folder	
.classpath	11/30/2022 10:18 PM	CLASSPATH File	1 KB
.gitattributes	11/30/2022 1:13 PM	Text Document	1 KB
.project	11/30/2022 10:20 PM	PROJECT File	1 KB
antlr-4.11.1-complete	11/30/2022 1:13 PM	Executable Jar File	3,465 KB
ck project specifications	11/30/2022 1:13 PM	Text Document	1 KB
Cypher	11/30/2022 1:13 PM	Executable Jar File	3 KB
jasmin	11/30/2022 1:13 PM	Executable Jar File	126 KB
name.class	12/1/2022 10:33 AM	CLASS File	2 KB
name.j	12/1/2022 10:35 AM	J File	2 KB
name2.class	12/1/2022 2:51 PM	CLASS File	2 KB
name2.j	12/1/2022 2:51 PM	J File	3 KB
README	11/30/2022 1:13 PM	Markdown Source...	1 KB
sample.ck	12/1/2022 10:30 AM	CK File	1 KB
sample2.ck	12/1/2022 2:55 PM	CK File	1 KB
simple.class	12/1/2022 2:59 PM	CLASS File	2 KB
simple.j	12/1/2022 2:55 PM	J File	3 KB

```
PASS 1 Syntax: There were no syntax errors.

PASS 2 Semantics:

===== CROSS-REFERENCE TABLE =====

*** PROGRAM simple ***

Identifier       Line numbers    Type specification
----------       ------------    ------------------
i                009 010 015 016
                                 Kind: variable
                                 Scope nesting level: 1
                                 Type form: scalar, Type id: integer
plain            006 007
                                 Kind: variable
                                 Scope nesting level: 1
                                 Type form: scalar, Type id: string
x                012 013
                                 Kind: variable
                                 Scope nesting level: 1
                                 Type form: scalar, Type id: real

PASS 3 Compilation: Object file "simple.j" created.
```

Simple Program
- Assembly, Execution

Windows PowerShell

```
PS C:\Users\jwebb> cd 'C:\Users\jwebb\OneDrive\Desktop\College\2022-2023 (Grad 1)\CS 153\Assignments\Final Project\CS153-Final-main-3\CS153-Final-main-3'
PS C:\Users\jwebb\OneDrive\Desktop\College\2022-2023 (Grad 1)\CS 153\Assignments\Final Project\CS153-Final-main-3\CS153-Final-main-3> java -jar jasmin.jar simple.j
Generated: simple.class
PS C:\Users\jwebb\OneDrive\Desktop\College\2022-2023 (Grad 1)\CS 153\Assignments\Final Project\CS153-Final-main-3\CS153-Final-main-3> java simple
hellotest
[1 milliseconds execution time.]
PS C:\Users\jwebb\OneDrive\Desktop\College\2022-2023 (Grad 1)\CS 153\Assignments\Final Project\CS153-Final-main-3\CS153-Final-main-3>
```

Live Demo!