COMPILERS DESIGN

ASSIGNMENT-1

1. GROUP MEMBERS

Pradeep Ambavaram 14079
Ananth Bhukya 14183
Ankush singh 150107

3. Compiler Specifications

Source language : JavaScript
Implementation language : Python
Target language : Assembly

BNF For EcmaScript.jj
NON-TERMINALS

PrimaryExpression	::=	"this"
	|	ObjectLiteral
	|	("(" Expression ")")
	|	Identifier
	|	ArrayLiteral
	|	Literal
Literal	::=	(<DECIMAL_LITERAL> | <HEX_INTEGER_LITERAL> | <STRING_LITERAL> | <BOOLEAN_LITERAL> | <NULL_LITERAL> | <REGULAR_EXPRESSION_LITERAL>)
Identifier	::=	<IDENTIFIER_NAME>
ArrayLiteral	::=	"[" ((Elision)? "]" | ElementList Elision "]" | (ElementList)? "]")
ElementList	::=	(Elision)? AssignmentExpression (Elision AssignmentExpression)*

Elision	::=	(",")+
ObjectLiteral	::=	"{" (PropertyNameAndValueList)? "}"
PropertyNameAndValueList	::=	PropertyNameAndValue ("," PropertyNameAndValue | ",")*
PropertyNameAndValue	::=	PropertyName ":" AssignmentExpression
PropertyName	::=	Identifier
	|	<STRING_LITERAL>
	|	<DECIMAL_LITERAL>

MemberExpression	::=	((FunctionExpression | PrimaryExpression) (MemberExpressionPart)*)
	|	AllocationExpression
MemberExpressionForIn	::=	((FunctionExpression | PrimaryExpression) (MemberExpressionPart)*)
AllocationExpression	::=	("new" MemberExpression ((Arguments (MemberExpressionPart)*)*))
MemberExpressionPart	::=	("[" Expression "]")
	|	("." Identifier)
CallExpression	::=	MemberExpression Arguments (CallExpressionPart)*
CallExpressionForIn	::=	MemberExpressionForIn Arguments (CallExpressionPart)*

CallExpressionPart	: : =	Arguments
	|	("[" Expression "]")
	|	("." Identifier)
Arguments	: : =	"(" (ArgumentList)? ")"
ArgumentList	: : =	AssignmentExpression ("," AssignmentExpression)*
LeftHandSideExpression	: : =	CallExpression
	|	MemberExpression
LeftHandSideExpression ForIn	: : =	CallExpressionForIn
	|	MemberExpressionForIn
PostfixExpression	: : =	LeftHandSideExpression (PostfixOperator)?
PostfixOperator	: : =	("++" | "--")
UnaryExpression	: : =	(PostfixExpression | (UnaryOperator UnaryExpression)+)
UnaryOperator	: : =	("delete" | "void" | "typeof" | "++" | "--" | "+" | "-" | "~" | "!")
MultiplicativeExpression	: :	UnaryExpression (MultiplicativeOperator UnaryExpression)*

	=	
MultiplicativeOperator	: : =	("*" | <SLASH> | "%")
AdditiveExpression	: : =	MultiplicativeExpression (AdditiveOperator MultiplicativeExpression)*
AdditiveOperator	: : =	("+" | "-")
ShiftExpression	: : =	AdditiveExpression (ShiftOperator AdditiveExpression)*
ShiftOperator	: : =	("<<" | ">>" | ">>>")
RelationalExpression	: : =	ShiftExpression (RelationalOperator ShiftExpression)*
RelationalOperator	: : =	("<" | ">" | "<=" | ">=" | "instanceof" | "in")
RelationalExpressionNoIn	: : =	ShiftExpression (RelationalNoInOperator ShiftExpression)*
RelationalNoInOperator	: : =	("<" | ">" | "<=" | ">=" | "instanceof")
EqualityExpression	: : =	RelationalExpression (EqualityOperator RelationalExpression)*
EqualityExpressionNoIn	: : =	RelationalExpressionNoIn (EqualityOperator RelationalExpressionNoIn)*

EqualityOperator	::=	("==" | "!=" | "===" | "!==")
BitwiseANDExpression	::=	EqualityExpression (BitwiseANDOperator EqualityExpression)*
BitwiseANDExpressionNoIn	::=	EqualityExpressionNoIn (BitwiseANDOperator EqualityExpressionNoIn)*
BitwiseANDOperator	::=	"&"
BitwiseXORExpression	::=	BitwiseANDExpression (BitwiseXOROperator BitwiseANDExpression)*
BitwiseXORExpressionNoIn	::=	BitwiseANDExpressionNoIn (BitwiseXOROperator BitwiseANDExpressionNoIn)*
BitwiseXOROperator	::=	"^"
BitwiseORExpression	::=	BitwiseXORExpression (BitwiseOROperator BitwiseXORExpression)*
BitwiseORExpressionNoIn	::=	BitwiseXORExpressionNoIn (BitwiseOROperator BitwiseXORExpres.sionNoIn)*
BitwiseOROperator	::=	"|"
LogicalANDExpression	::=	BitwiseORExpression (LogicalANDOperator BitwiseORExpression)*
LogicalANDExpressionN	:	BitwiseORExpressionNoIn (LogicalANDOperator

oIn	: =	BitwiseORExpressionNoIn)*
LogicalANDOperator	: : =	"&&"
LogicalORExpression	: : =	LogicalANDExpression (LogicalOROperator LogicalANDExpression)*
LogicalORExpressionNoIn	: : =	LogicalANDExpressionNoIn (LogicalOROperator LogicalANDExpressionNoIn)*
LogicalOROperator	: : =	"||"
ConditionalExpression	: : =	LogicalORExpression ("?" AssignmentExpression ":" AssignmentExpression)?
ConditionalExpressionNoIn	: : =	LogicalORExpressionNoIn ("?" AssignmentExpression ":" AssignmentExpressionNoIn)?
AssignmentExpression	: : =	(LeftHandSideExpression AssignmentOperator AssignmentExpression | ConditionalExpression)
AssignmentExpressionNoIn	: : =	(LeftHandSideExpression AssignmentOperator AssignmentExpressionNoIn | ConditionalExpressionNoIn)
AssignmentOperator	: : =	("=" | "*=" | <SLASHASSIGN> | "%=" | "+=" | "-=" | "<<=" | ">>=" | ">>>=" | "&=" | "^=" | "|=")
Expression	: : =	AssignmentExpression ("," AssignmentExpression)*
ExpressionNoIn	: :	AssignmentExpressionNoIn ("," AssignmentExpressionNoIn)*

```
                        =
                        :
Statement               :   Block
                        =
                        |   JScriptVarStatement
                        |   VariableStatement
                        |   EmptyStatement
                        |   LabelledStatement
                        |   ExpressionStatement
                        |   IfStatement
                        |   IterationStatement
                        |   ContinueStatement
                        |   BreakStatement
                        |   ImportStatement
                        |   ReturnStatement
                        |   WithStatement
                        |   SwitchStatement
                        |   ThrowStatement
                        |   TryStatement
                        :
Block                   :   "{" ( StatementList )? "}"
                        =
                        :
StatementList           :   ( Statement )+
                        =
                        :
VariableStatement       :   "var" VariableDeclarationList ( ";" )?
                        =
```

VariableDeclarationList	: : =	VariableDeclaration ("," VariableDeclaration)*
VariableDeclarationListNoIn	: : =	VariableDeclarationNoIn ("," VariableDeclarationNoIn)*
VariableDeclaration	: : =	Identifier (Initialiser)?
VariableDeclarationNoIn	: : =	Identifier (InitialiserNoIn)?
Initialiser	: : =	"=" AssignmentExpression
InitialiserNoIn	: : =	"=" AssignmentExpressionNoIn
EmptyStatement	: : =	";"
ExpressionStatement	: : =	Expression (";")?
IfStatement	: : =	"if" "(" Expression ")" Statement ("else" Statement)?
IterationStatement	: : =	("do" Statement "while" "(" Expression ")" (";")?)
	|	("while" "(" Expression ")" Statement)
	|	("for" "(" (ExpressionNoIn)? ";" (Expression)? ";" (Expression)? ")" Statement)
	|	("for" "(" "var" VariableDeclarationList ";" (Expression)? ";" (

```
                                Expression )? ")" Statement )
                        |       ( "for" "(" "var" VariableDeclarationNoIn "in" Expression ")" Statement
                                )
                        |       ( "for" "(" LeftHandSideExpressionForIn "in" Expression ")" Statement
                                )
                        ;
ContinueStatement       :       "continue" ( Identifier )? ( ";" )?
                        =
                        ;
BreakStatement          :       "break" ( Identifier )? ( ";" )?
                        =
                        ;
ReturnStatement         :       "return" ( Expression )? ( ";" )?
                        =
                        ;
WithStatement           :       "with" "(" Expression ")" Statement
                        =
                        ;
SwitchStatement         :       "switch" "(" Expression ")" CaseBlock
                        =
                        ;
CaseBlock               :       "{" ( CaseClauses )? ( "}" | DefaultClause ( CaseClauses )? "}" )
                        =
                        ;
CaseClauses             :       ( CaseClause )+
                        =
                        ;
CaseClause              :       ( ( "case" Expression ":" ) ) ( StatementList )?
                        =
                        ;
DefaultClause           :       ( ( "default" ":" ) ) ( StatementList )?
                        =
                        ;
LabelledStatement       :       Identifier ":" Statement
                        ;
```

```
                        =

                        :
ThrowStatement          :   "throw" Expression ( ";" )?
                        =

                        :
TryStatement            :   "try" Block ( ( Finally | Catch ( Finally )? ) )
                        =

                        :
Catch                   :   "catch" "(" Identifier ")" Block
                        =

                        :
Finally                 :   "finally" Block
                        =

                        :
FunctionDeclaration     :   "function" Identifier ( "(" ( FormalParameterList )? ")" ) FunctionBody
                        =

                        :
FunctionExpression      :   "function" ( Identifier )? ( "(" ( FormalParameterList )? ")" )
                        :   FunctionBody
                        =

                        :
FormalParameterList     :   Identifier ( "," Identifier )*
                        =

                        :
FunctionBody            :   "{" ( SourceElements )? "}"
                        =

                        :
Program                 :   ( SourceElements )? <EOF>
                        =

                        :
SourceElements          :   ( SourceElement )+
                        =

                        :
SourceElement           :   FunctionDeclaration
                        =
```

| Statement

ImportStatement : : = "import" Name ("." "*")? ";"

Name : : = <IDENTIFIER_NAME> ("." <IDENTIFIER_NAME>)*

JScriptVarStatement : : = "var" JScriptVarDeclarationList (";")?

JScriptVarDeclarationList : : = JScriptVarDeclaration ("," JScriptVarDeclaration)*

JScriptVarDeclaration : : = Identifier ":" <IDENTIFIER_NAME> (Initialiser)?

insertSemiColon : : = *java code*

Source:
https://tomcopeland.blogs.com/ecmascript.html?fbclid=IwAR06rtTC6ac7IsWPO9eFN85TGAvbiMYPF9RUpmvSWrFmL2MjKfCjebzlBgo