Indian Institute of Technology Kanpur
CS335: Compiler Design, 18–19

ASSIGNMENT 0

Student Name: Prann Bansal, Manish Kumar Bera, Gurpreet Singh
Roll Number: 150381, 150510, 150259
Date: 21 January, 2018

Team Members

Names	**Roll Numbers**	**Email**
Manish Kumar Bera	150381	mkbera@iitk.ac.in
Prann Bansal	150510	prann@iitk.ac.in
Gurpreet Singh	150259	guggu@iitk.ac.in

T-Diagram

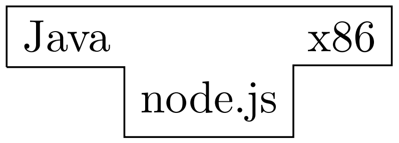

Figure 1: T-Diagram for the proposed compiler

The T-Diagram for the proposed compiler is given in Figure 1. We aim to build a compiler for a subset of the Java programming language, compiled to the assembly language x86, on a Node.js platform.

BNF

Below, we provide the subset of the Java BNF obtained from — https://users-cs.au.dk/amoeller/dRegAut/JavaBNF.html

1. **Programs.**

CONSTRUCT		RULES
`<compilation unit>`	::=	`<import declarations>? <type declarations>?`

2. **Declarations.**

CONSTRUCT		RULES
`<import declarations>`	::=	`<import declaration> | <import declarations> <import declaration>`
`<import declaration>`	::=	`<single type import declaration> | <type import on demand declaration>`
`<single type import declaration>`	::=	`import <type name> ;`
`<type declarations>`	::=	`<type declaration> | <type declarations> <type declaration>`
`<type declaration>`	::=	`<class declaration> | ;`
`<class declaration>`	::=	`<class modifiers>? class <identifier> <super>? <class body>`
`<class modifiers>`	::=	`<class modifier> | <class modifiers> <class modifier>`
`<class modifier>`	::=	`public`
`<super>`	::=	`extends <class type>`
`<class body>`	::=	`<class body declarations>?`
`<class body declarations>`	::=	`<class body declaration> | <class body declarations> <class body declaration>`
`<class body declaration>`	::=	`<class member declaration> | <static initializer> | <constructor declaration>`
`<class member declaration>`	::=	`<field declaration> | <method declaration>`
`<static initializer>`	::=	`static <block>`

```
       <constructor declaration>  : : =  <constructor modifiers>?  <constructor declarator> <constructor body>
         <constructor modifiers>  : : =  <constructor modifier> | <constructor modifiers> <constructor modifier>
          <constructor modifier>  : : =  public
        <constructor declarator>  : : =  <type name> ( <formal parameter list>?  )
         <formal parameter list>  : : =  <formal parameter> | <formal parameter list> , <formal parameter>
              <formal parameter>  : : =  <type> <variable declarator id>
              <constructor body>  : : =   <explicit constructor invocation>?  <block statements>?
<explicit constructor invocation>  : : =  this ( <argument list>?  )  | super ( <argument list>?  )
             <field declaration>  : : =  <field modifiers>?  <type> <variable declarators> ;
               <field modifiers>  : : =  <field modifier> | <field modifiers> <field modifier>
                <field modifier>  : : =  public | static
          <variable declarators>  : : =  <variable declarator> | <variable declarators> , <variable declarator>
           <variable declarator>  : : =  <variable declarator id> | <variable declarator id> = <variable
                                         initializer>
        <variable declarator id>  : : =  <identifier> | <variable declarator id> [ ]
          <variable initializer>  : : =  <expression> | <array initializer>
            <method declaration>  : : =  <method header> <method body>
                 <method header>  : : =  <method modifiers>?  <result type> <method declarator>
                   <result type>  : : =  <type> | void
              <method modifiers>  : : =  <method modifier> | <method modifiers> <method modifier>
               <method modifier>  : : =  public | static
             <method declarator>  : : =  <identifier> ( <formal parameter list>?  )
                   <method body>  : : =  <block> | ;
          <constant declaration>  : : =  <constant modifiers> <type> <variable declarator>
```

<constant modifiers>	: : =	public | static
<array initializer>	: : =	<variable initializers>? , ?
<variable initializers>	: : =	<variable initializer> | <variable initializers> , <variable initializer>
<variable initializer>	: : =	<expression> | <array initializer>

3. Types.

CONSTRUCT		RULES
<block>	: : =	<block statements>?
<block statements>	: : =	<block statement> | <block statements> <block statement>
<block statement>	: : =	<local variable declaration statement> | <statement>
<local variable declaration statement>	: : =	<local variable declaration> ;
<local variable declaration>	: : =	<type> <variable declarators>
<statement>	: : =	<statement without trailing substatement> | <if then statement> | <if then else statement> | <while statement> | <for statement>
<statement no short if>	: : =	<statement without trailing substatement> | <if then else statement no short if> | <while statement no short if> | <for statement no short if>
<statement without trailing substatement>	: : =	<block> | <empty statement> | <expression statement> | <switch statement> | <do statement> | <break statement> | <continue statement> | <return statement>
<empty statement>	: : =	;
<expression statement>	: : =	<statement expression> ;
<statement expression>	: : =	<assignment> | <preincrement expression> | <postincrement expression> | <predecrement expression> | <postdecrement expression> | <method invocation> | <class instance creation expression>
<if then statement>	: : =	if (<expression>) <statement>
<if then else statement>	: : =	if (<expression>) <statement no short if> else <statement>

<if then else statement no short if>	: : =	if (<expression>) <statement no short if> else <statement no short if>
<switch statement>	: : =	switch (<expression>) <switch block>
<switch block>	: : =	<switch block statement groups>? <switch labels>?
<switch block statement groups>	: : =	<switch block statement group> | <switch block statement groups> <switch block statement group>
<switch block statement group>	: : =	<switch labels> <block statements>
<switch labels>	: : =	<switch label> | <switch labels> <switch label>
<switch label>	: : =	case <constant expression> : | default :
<while statement>	: : =	while (<expression>) <statement>
<while statement no short if>	: : =	while (<expression>) <statement no short if>
<do statement>	: : =	do <statement> while (<expression>) ;
<for statement>	: : =	for (<for init>? ; <expression>? ; <for update>?) <statement>
<for statement no short if>	: : =	for (<for init>? ; <expression>? ; <for update>?) <statement no short if>
<for init>	: : =	<statement expression list> | <local variable declaration>
<for update>	: : =	<statement expression list>
<statement expression list>	: : =	<statement expression> | <statement expression list> , <statement expression>
<break statement>	: : =	break ;
<continue statement>	: : =	continue ;
<return statement>	: : =	return <expression>? ;

4. Blocks and Commands.

CONSTRUCT		RULES
<constant expression>	: : =	<expression>

<expression>	: : =	<assignment expression>
<assignment expression>	: : =	<conditional expression> | <assignment>
<assignment>	: : =	<left hand side> <assignment operator> <assignment expression>
<left hand side>	: : =	<expression name> | <field access> | <array access>
<assignment operator>	: : =	= | *= | /= | %= | += | -= | «= | »= | &= | ∧= | |=
<conditional expression>	: : =	<conditional or expression> | <conditional or expression> ? <expression> : <conditional expression>
<conditional or expression>	: : =	<conditional and expression> | <conditional or expression> || <conditional and expression>
<conditional and expression>	: : =	<inclusive or expression> | <conditional and expression> && <inclusive or expression>
<inclusive or expression>	: : =	<exclusive or expression> | <inclusive or expression> #| <exclusive or expression>
<exclusive or expression>	: : =	<and expression> | <exclusive or expression> ∧ <and expression>
<and expression>	: : =	<equality expression> | <and expression> & <equality expression>
<equality expression>	: : =	<relational expression> | <equality expression> == <relational expression> | <equality expression> != <relational expression>
<relational expression>	: : =	<shift expression> | <relational expression> < <shift expression> | <relational expression> > <shift expression> | <relational expression> <= <shift expression> | <relational expression> >= <shift expression> | <relational expression> instanceof <reference type>
<shift expression>	: : =	<additive expression> | <shift expression> « <additive expression> | <shift expression> » <additive expression>
<additive expression>	: : =	<multiplicative expression> | <additive expression> + <multiplicative expression> | <additive expression> - <multiplicative expression>
<multiplicative expression>	: : =	<unary expression> | <multiplicative expression> * <unary expression> | <multiplicative expression> / <unary expression> | <multiplicative expression> % <unary expression>

<cast expression>	: : =	(<primitive type>) <unary expression> | (<reference type>) <unary expression not plus minus>
<unary expression>	: : =	<preincrement expression> | <predecrement expression> | + <unary expression> | - <unary expression> | <unary expression not plus minus>
<predecrement expression>	: : =	- <unary expression>
<preincrement expression>	: : =	++ <unary expression>
<unary expression not plus minus>	: : =	<postfix expression> | <unary expression> | ! <unary expression> | <cast expression>
<postdecrement expression>	: : =	<postfix expression> -
<postincrement expression>	: : =	<postfix expression> ++
<postfix expression>	: : =	<primary> | <expression name> | <postincrement expression> | <postdecrement expression>
<method invocation>	: : =	<method name> (<argument list>?) | <primary> . <identifier> (<argument list>?) | super . <identifier> (<argument list>?)
<field access>	: : =	<primary> . <identifier> | super . <identifier>
<primary>	: : =	<primary no new array> | <array creation expression>
<primary no new array>	: : =	<literal> | this | (<expression>) | <class instance creation expression> | <field access> | <method invocation> | <array access>
<class instance creation expression>	: : =	new <class type> (<argument list>?)
<argument list>	: : =	<expression> | <argument list> , <expression>
<array creation expression>	: : =	new <primitive type> <dim exprs> <dims>? | new <class or interface type> <dim exprs> <dims>?
<dim exprs>	: : =	<dim expr> | <dim exprs> <dim expr>
<dim expr>	: : =	[<expression>]
<dims>	: : =	[ ] | <dims> [ ]

<array access>	: : =	<expression name> [<expression>] | <primary no new array> [<expression>]

5. Expressions.

CONSTRUCT		RULES
<constant expression>	: : =	<expression>
<expression>	: : =	<assignment expression>
<assignment expression>	: : =	<conditional expression> | <assignment>
<assignment>	: : =	<left hand side> <assignment operator> <assignment expression>
<left hand side>	: : =	<expression name> | <field access> | <array access>
<assignment operator>	: : =	= | *= | /= | %= | += | -= | «= | »= | &= | ∧= | |=
<conditional expression>	: : =	<conditional or expression> | <conditional or expression> ? <expression> : <conditional expression>
<conditional or expression>	: : =	<conditional and expression> | <conditional or expression> || <conditional and expression>
<conditional and expression>	: : =	<inclusive or expression> | <conditional and expression> && <inclusive or expression>
<inclusive or expression>	: : =	<exclusive or expression> | <inclusive or expression> | <exclusive or expression>
<exclusive or expression>	: : =	<and expression> | <exclusive or expression> ∧ <and expression>
<and expression>	: : =	<equality expression> | <and expression> & <equality expression>
<equality expression>	: : =	<relational expression> | <equality expression> == <relational expression> | <equality expression> != <relational expression>
<relational expression>	: : =	<shift expression> | <relational expression> < <shift expression> | <relational expression> > <shift expression> | <relational expression> <= <shift expression> | <relational expression> >= <shift expression> | <relational expression> instanceof <reference type>

<shift expression>	: : =	<additive expression> | <shift expression> « <additive expression> | <shift expression> » <additive expression>
<additive expression>	: : =	<multiplicative expression> | <additive expression> + <multiplicative expression> | <additive expression> - <multiplicative expression>
<multiplicative expression>	: : =	<unary expression> | <multiplicative expression> * <unary expression> | <multiplicative expression> / <unary expression> | <multiplicative expression> % <unary expression>
<cast expression>	: : =	(<primitive type>) <unary expression> | (<reference type>) <unary expression not plus minus>
<unary expression>	: : =	<preincrement expression> | <predecrement expression> | + <unary expression> | - <unary expression> | <unary expression not plus minus>
<predecrement expression>	: : =	- <unary expression>
<preincrement expression>	: : =	++ <unary expression>
<unary expression not plus minus>	: : =	<postfix expression> | <unary expression> | ! <unary expression> | <cast expression>
<postdecrement expression>	: : =	<postfix expression> -
<postincrement expression>	: : =	<postfix expression> ++
<postfix expression>	: : =	<primary> | <expression name> | <postincrement expression> | <postdecrement expression>
<method invocation>	: : =	<method name> (<argument list>?) | <primary> . <identifier> (<argument list>?) | super . <identifier> (<argument list>?)
<field access>	: : =	<primary> . <identifier> | super . <identifier>
<primary>	: : =	<primary no new array> | <array creation expression>
<primary no new array>	: : =	<literal> | this | (<expression>) | <class instance creation expression> | <field access> | <method invocation> | <array access>
<class instance creation expression>	: : =	new <class type> (<argument list>?)
<argument list>	: : =	<expression> | <argument list> , <expression>

<array creation expression>	::=	new <primitive type> <dim exprs> <dims>? | new <class or interface type> <dim exprs> <dims>?
<dim exprs>	::=	<dim expr> | <dim exprs> <dim expr>
<dim expr>	::=	[<expression>]
<dims>	::=	[ ] | <dims> [ ]
<array access>	::=	<expression name> [<expression>] | <primary no new array> [<expression>]

6. Tokens.

CONSTRUCT		RULES
<type name>	::=	<identifier>
<expression name>	::=	<identifier> | <ambiguous name> . <identifier>
<method name>	::=	<identifier> | <ambiguous name>. <identifier>
<ambiguous name>	::=	<identifier> | <ambiguous name>. <identifier>
<literal>	::=	<integer literal> | <floating-point literal> | <boolean literal> | <character literal> | <string literal> | <null literal>
<integer literal>	::=	0 | <non zero digit> <digits>?
<digits>	::=	<digit> | <digits> <digit>
<digit>	::=	0 | <non zero digit>
<non zero digit>	::=	1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<floating-point literal>	::=	<digits> . <digits>?
<signed integer>	::=	<sign>? <digits>
<sign>	::=	+ | -
<boolean literal>	::=	true | false
<character literal>	::=	' <single character> ' | ' <escape sequence> '

<single character>	::=	<input character> except ' and \
<string literal>	::=	" <string characters>? "
<string characters>	::=	<string character> | <string characters> <string character>
<string character>	::=	<input character> except " and \ | <escape character>
<null literal>	::=	null
<keyword>	::=	boolean | break | byte | case | char | class | const | continue | default | do | double | else | extends | float | for | if | import | instanceof | int | long | new | return | short | static | super | switch | this | void | while

Deleted Constructs

1. **Programs.**

CONSTRUCT		RULES
<compilation unit>	::−	<package declaration>? <import declarations>? <type declarations>?
	::+	<import declarations>? <type declarations>?

2. **Declarations.**

CONSTRUCT		RULES
<package declaration>	::−	package <package name> ;
<type import on demand declaration>	::−	import <package name> . * ;
<type declaration>	::−	<class declaration> | <interface declaration> | ;
	::+	<class declaration> | ;
<class declaration>	::−	<class modifiers>? class <identifier> <super>? <interfaces>? <class body>
	::+	<class modifiers>? class <identifier> <super>? <class body>

```
          <class modifier>  ::−  public | abstract | final
                            ::+  public

              <interfaces>  ::−  implements <interface type list>

     <interface type list>  ::−  <interface type> | <interface type list> , <interface type>

                                 <constructor modifiers>?  <constructor declarator> <throws>?  <constructor
<constructor declaration>   ::−  body>
                            ::+  <constructor modifiers>?  <constructor declarator> <constructor body>

    <constructor modifier>  ::−  public | protected | private
                            ::+  public

  <constructor declarator>  ::−  <simple type name> ( <formal parameter list>?  )
                            ::+  <type name> ( <formal parameter list>?  )

                  <throws>  ::−  throws <class type list>

         <class type list>  ::−  <class type> | <class type list> , <class type>

          <field modifier>  ::−  public | protected | private | static | final | transient | volatile
                            ::+  public | static

           <method header>  ::−  <method modifiers>?  <result type> <method declarator> <throws>?
                            ::+  <method modifiers>?  <result type> <method declarator>

                                 public | protected | private | static | abstract | final | synchronized |
         <method modifier>  ::−  native
                            ::+  public | static

                                 <interface modifiers>?  interface <identifier> <extends interfaces>?
   <interface declaration>  ::−  <interface body>

     <interface modifiers>  ::−  <interface modifier> | <interface modifiers> <interface modifier>
```

<interface modifier>	::−	public | abstract
<extends interfaces>	::−	extends <interface type> | <extends interfaces> , <interface type>
<interface body>	::−	{ <interface member declarations>? }
<interface member declarations>	::−	<interface member declaration> | <interface member declarations> <interface member declaration>
<interface member declaration>	::−	<constant declaration> | <abstract method declaration>
<constant modifiers>	::−	public | static | final
	::+	public | static
<abstract method declaration>	::−	<abstract method modifiers>? <result type> <method declarator> <throws>? ;
<abstract method modifiers>	::−	<abstract method modifier> | <abstract method modifiers> <abstract method modifier>
<abstract method modifier>	::−	public | abstract

3. **Blocks and Commands.**

CONSTRUCT		RULES
<statement>	::−	<statement without trailing substatement> | <labeled statement> | <if then statement> | <if then else statement> | <while statement> | <for statement>
	::+	<statement without trailing substatement> | <if then statement> | <if then else statement> | <while statement> | <for statement>
<statement no short if>	::−	<statement without trailing substatement> | <labeled statement no short if> | <if then else statement no short if> | <while statement no short if> | <for statement no short if>
	::+	<statement without trailing substatement> | <if then else statement no short if> | <while statement no short if> | <for statement no short if>

<statement without trailing substatement>	: : −	<block> | <empty statement> | <expression statement> | <switch statement> | <do statement> | <break statement> | <continue statement> | <return statement> | <synchronized statement> | <throws statements> | <try statement>
	: : +	<block> | <empty statement> | <expression statement> | <switch statement> | <do statement> | <break statement> | <continue statement> | <return statement>
<labeled statement>	: : −	<identifier> : <statement>
<labeled statement no short if>	: : −	<identifier> : <statement no short if>
<break statement>	: : −	break <identifier>? ;
	: : +	break ;
<continue statement>	: : −	continue <identifier>? ;
	: : +	continue ;
<throws statement>	: : −	throw <expression> ;
<synchronized statement>	: : −	synchronized (<expression>) <block>
<try statement>	: : −	try <block> <catches> | try <block> <catches>? <finally>
<catches>	: : −	<catch clause> | <catches> <catch clause>
<catch clause>	: : −	catch (<formal parameter>) <block>
<finally >	: : −	finally <block>

4. Expressions.

CONSTRUCT		RULES
<assignment operator>	: : −	= | *= | /= | %= | += | -= | «= | »= | »>= | &= | ∧= | |=
	: : +	= | *= | /= | %= | += | -= | «= | »= | &= | ∧= | |=

<shift expression>	: : −	<additive expression> | <shift expression> « <additive expression> | <shift expression> » <additive expression> | <shift expression> »> <additive expression>
	: : +	<additive expression> | <shift expression> « <additive expression> | <shift expression> » <additive expression>

5. **Tokens.**

CONSTRUCT		RULES
<package name>	: : −	<identifier> | <package name> . <identifier>
<type name>	: : −	<identifier> | <package name> . <identifier>
	: : +	<identifier>
<simple type name>	: : −	<identifier>
<integer literal>	: : −	<decimal integer literal> | <hex integer literal> | <octal integer literal>
	: : +	0 | <non zero digit> <digits>?
<decimal integer literal>	: : −	<decimal numeral> <integer type suffix>?
<hex integer literal>	: : −	<hex numeral> <integer type suffix>?
<octal integer literal>	: : −	<octal numeral> <integer type suffix>?
<integer type suffix>	: : −	l | L
<decimal numeral>	: : −	0 | <non zero digit> <digits>?
<hex numeral>	: : −	0 x <hex digit> | 0 X <hex digit> | <hex numeral> <hex digit>
<hex digit>	: : −	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f | A | B | C | D | E | F
<octal numeral>	: : −	0 <octal digit> | <octal numeral> <octal digit>

```
             <octal digit>  ::−  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

  <floating-point literal>  ::−  <digits> . <digits>? <exponent part>? <float type suffix>?  | <digits>
                                 <exponent part>? <float type suffix>?
                            ::+  <digits> . <digits>?

           <exponent part>  ::−  <exponent indicator> <signed integer>

      <exponent indicator>  ::−  e | E

       <float type suffix>  ::−  f | F | d | D
          <string literal>  ::−  " <string characters>?"
                            ::+  " <string characters>?  "

                                 abstract | boolean | break | byte | case | catch | char | class | const |
                                 continue | default | do | double | else | extends | final | finally |
                                 float | for | goto | if | implements | import | instanceof | int |
                 <keyword>  ::−  interface | long | native | new | package | private | protected | public |
                                 return | short | static | super | switch | synchronized | this | throw |
                                 throws | transient | try | void | volatile | while
                                 boolean | break | byte | case | char | class | const | continue | default
                            ::+  | do | double | else | extends | float | for | if | import | instanceof |
                                 int | long | new | return | short | static | super | switch | this | void
                                 | while
```

Required Tools

1. **Lexer Generators.**

   a. jison-lex — https://www.npmjs.com/package/jison
   b. jacob — https://www.npmjs.com/package/jacob
   c. lexer — https://www.npmjs.com/package/lexer

2. **Parser Generators.**

a. jison — https://www.npmjs.com/package/jison
b. jacob — https://www.npmjs.com/package/jacob
c. pegjs — https://www.npmjs.com/package/pegjs