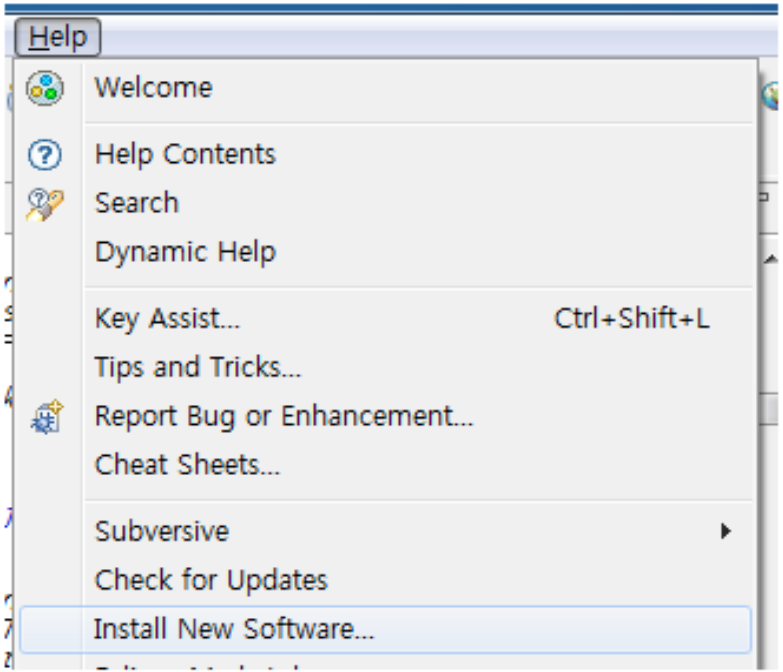
# Term Project

**Professor Joongheon Kim**

School of Computer Science and Engineering, Chung-Ang University, Seoul, Republic of Korea

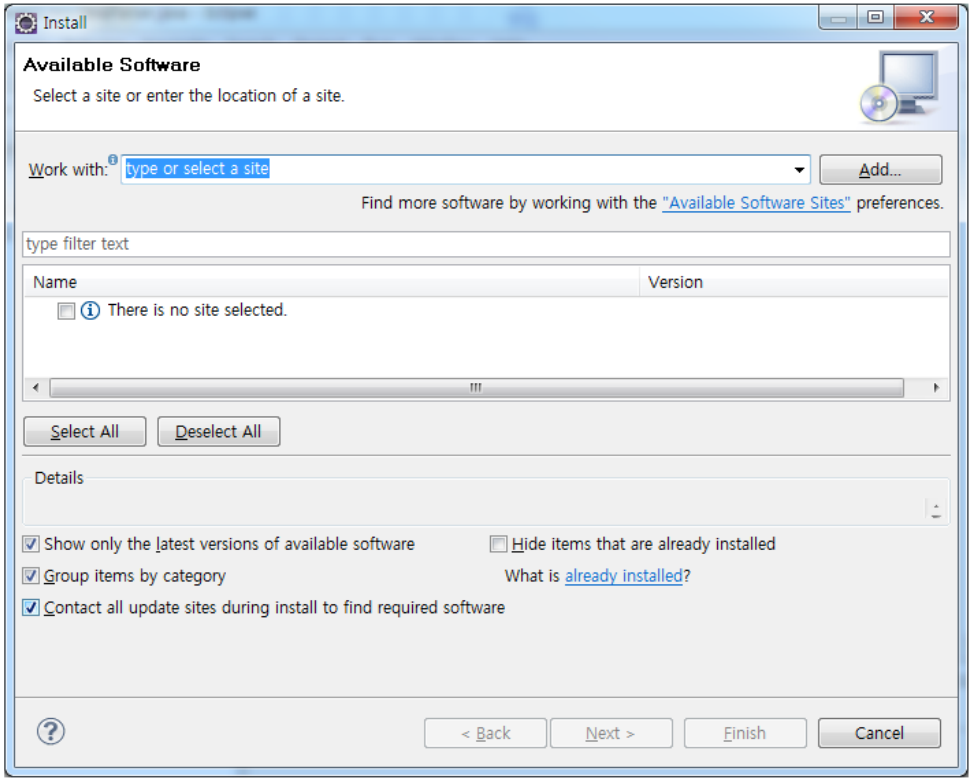
1. 메뉴 [Help] -> [Install New Software] 선택




**1. Open Eclipse**

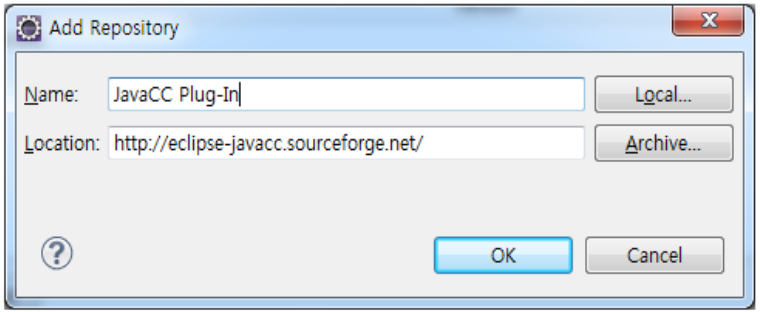
**2. Help -> Eclipse Marketplace**

2. [Install]창에서 [Add] 버튼 선택

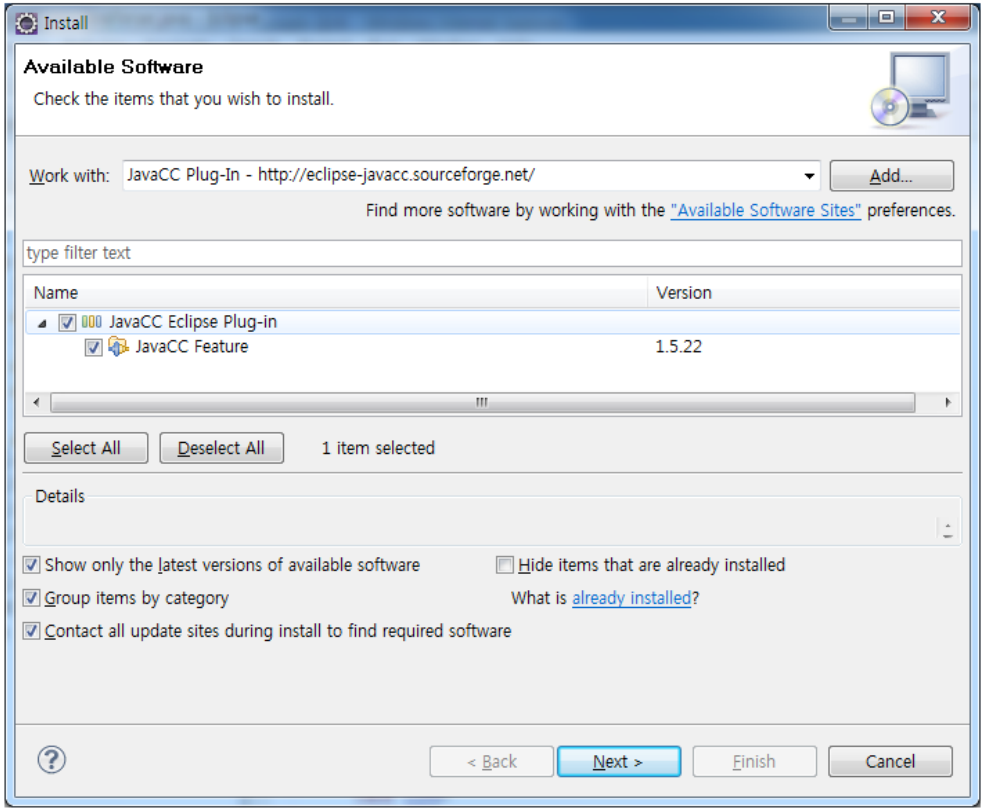



3. 아래와 같이 입력하고 [OK] 버튼 선택

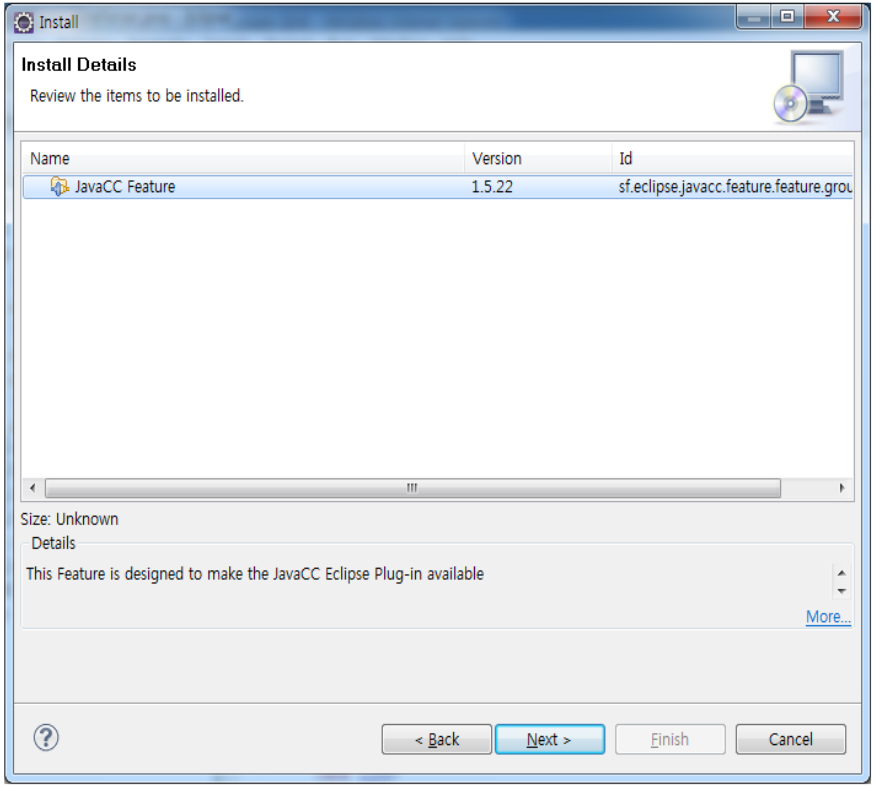
Name : JavaCC Plug-In
Location : http://eclipse-javacc.sourceforge.net/



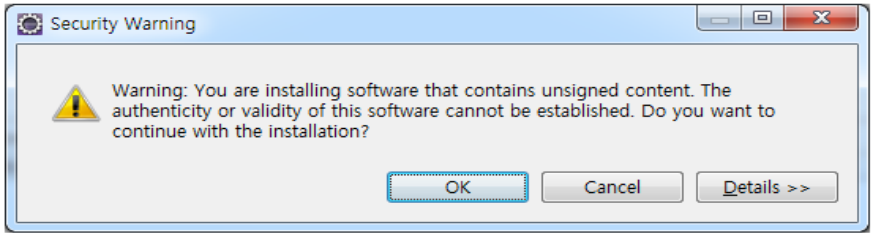
4. JavaCC Eclipse Plug-in 선택하고 [Next] 버튼 선택



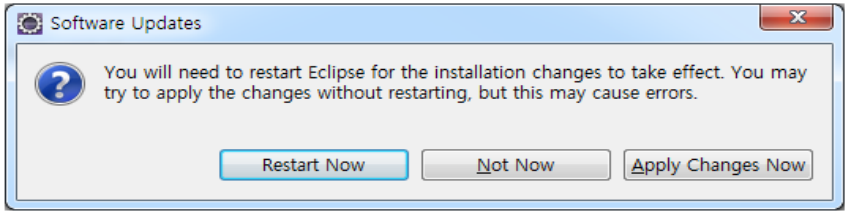
5. [Next] 버튼 선택

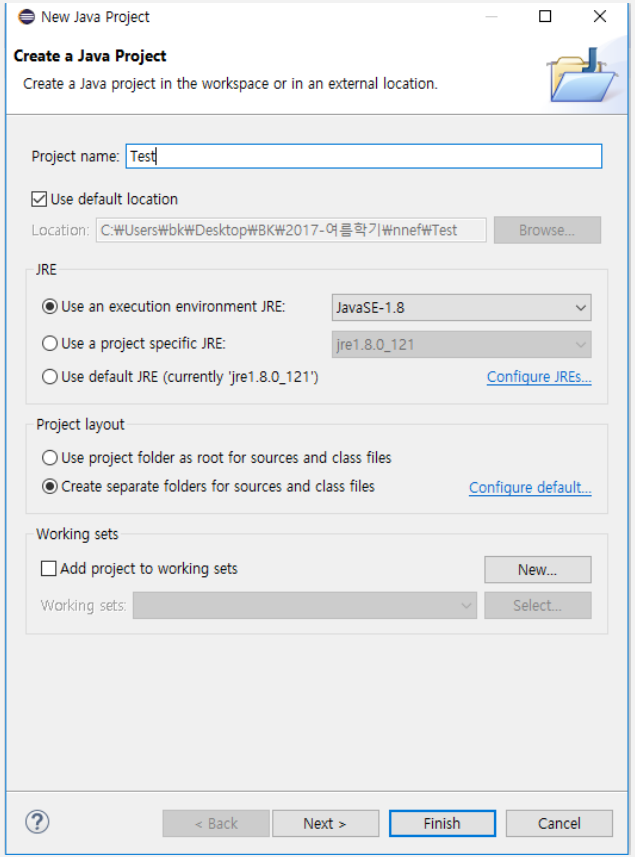



7. [OK] 버튼 선택



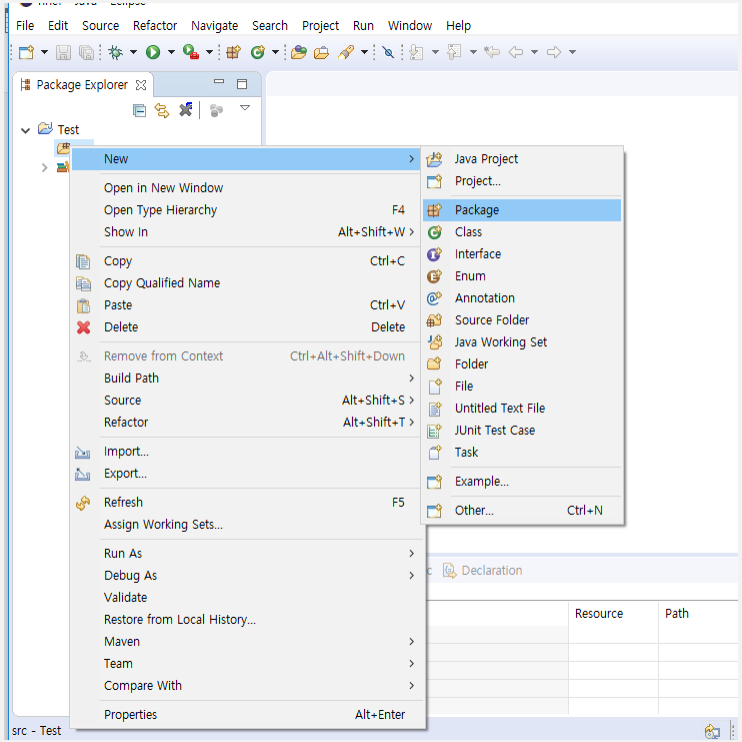

8. Plug-in 설치 완료 후, [Restar Now] 버튼을 선택하며 Eclipse을 다시 시작한다.





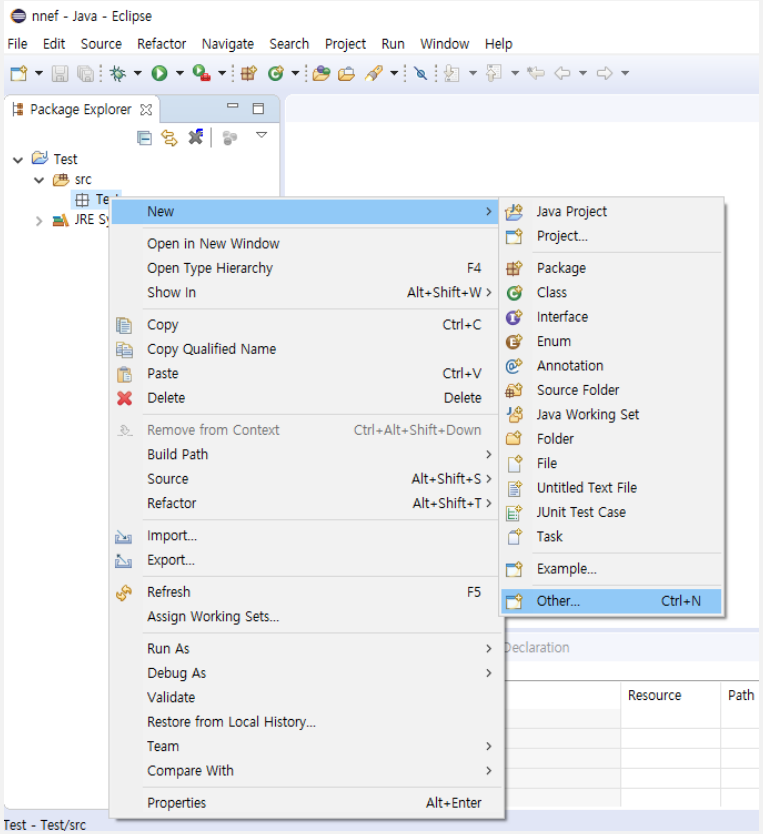

# 1. Create new Project

File -> New -> Java Project



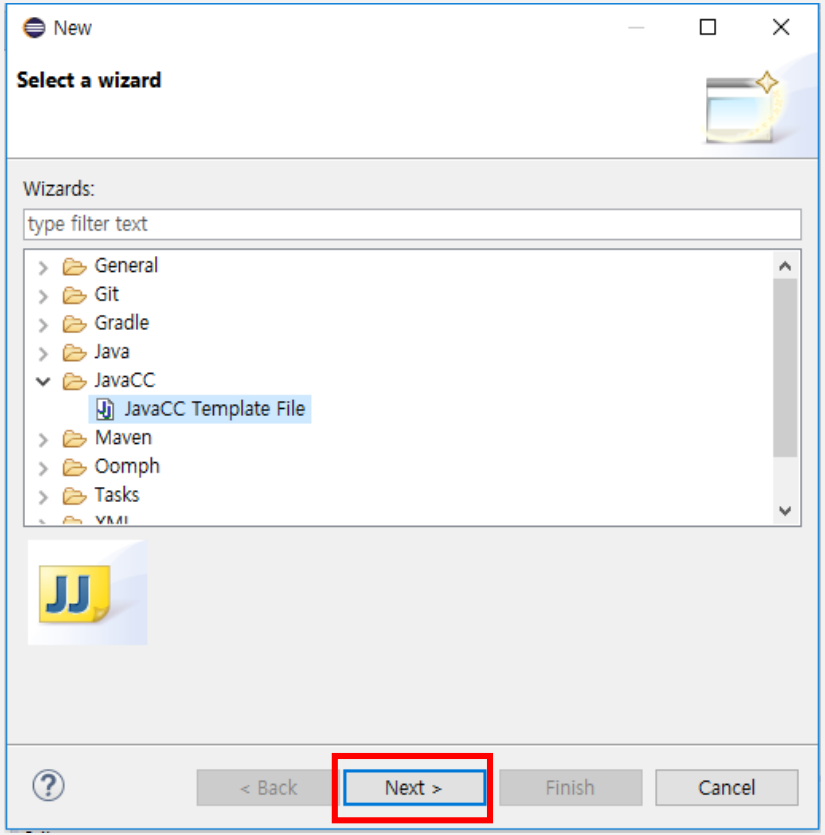

# 2. Create New Package

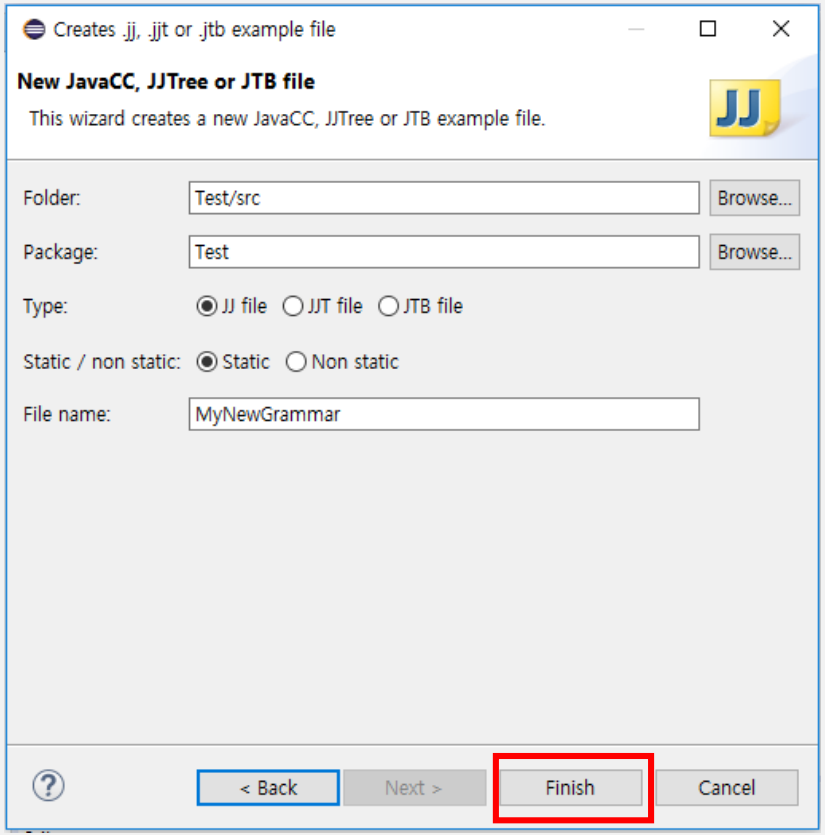
New -> Package




# 3. Create JavaCC File

New -> Other…-> JavaCC


New
Select a wizard
Wizards:
type filter text
General
Git
Gradle
Java
JavaCC
JavaCC Template File
Maven
Oomph
Tasks
< Back
Next >
Finish
Cancel


Creates .jj, .jjt or .jtb example file
New JavaCC, JJTree or JTB file
This wizard creates a new JavaCC, JJTree or JTB example file.
Folder:
Test/src
Browse...
Package:
Test
Browse...
Type:
JJ file
JJT file
JTB file
Static / non static:
Static
Non static
File name:
MyNewGrammar
< Back
Next >
Finish
Cancel

# <Obejctive>

(1) Interpret and make EBNF rules
(2) Top-down parser based on JavaCC
(3) Revise Left-Recursion EBNF rules

# <Requirement>

(1) Configure & Revise rules based on the given rules.
(2) Your parser must interpret the given examples and change it to the target code examples.

# [Basic Token]

If you need more Token definitions for your parser, you can create additional tokens.

<identifier>

An identifier is an alphanumeric sequence of ASCII characters that may also contain the underscore character. More specifically, identifiers **must** consist of the following ASCII characters: _, [a-z], [A-Z], [0-9]. The identifier **must not** start with a digit.

<numeric-literal>

A numeric literal consists of an integer part, an optional decimal point (.) and a fractional part, an e or E and an optionally signed integer exponent. The integer, fractional and the exponent parts **must** each consist of a sequence of decimal (base ten) digits ([0-9]). In case of flat syntax, the literal may be preceded by an optional - (minus) sign. In case of compositional syntax, the unary minus operator is used to enable negative numeric values, hence the - sign is not allowed.

<string-literal>

A string literal is a sequence of characters enclosed within ' or " characters. The end and start quotes **must** match. Any printable ASCII character may appear within the string, except for the start quote character, which **must** be escaped by the \ character. The \ character **must** also be escaped with the \ character.

# [Basic Token]

If you need more Token definitions for your parser, you can create additional tokens.

<logical-literal>

Logical literals are the values `true` and `false`.

<keyword>

The following alphabetic character sequences have special meaning with respect to the description syntax and thus **must not** be used as identifiers: `version`, `extension`, `graph`, `fragment`, `tensor`, `integer`, `scalar`, `logical`, `string`, `shape_of`, `length_of`, `range_of`, `for`, `in`, `yield`, `if`, `else`.

<operator>

The following character sequences have special meaning as operators in mathematical expressions: `+`, `-`, `*`, `/`, `^`, `<`, `<=`, `>`, `>=`, `==`, `!=`, `&&`, `||`, `!`.

# [SKIP]

If you need more SKIP definitions for your parser, you can create additional Skips.

(1)" " (white space)

(2)"□n" (new line)

(3)"□r□n" (newline)

(4)"□t" (tab)

# [Basic Rules]

If you need more rule definitions for your parser, you can create additional definitions.

```
<graph-definition> ::= <graph-declaration> <body>
<graph-declaration> ::= "graph" <identifier> "(" <identifier-list> ")"
                        "->" "(" <identifier-list> ")"
<identifier-list> ::= <identifier> ("," <identifier>)*

<body> ::= "{" <assignment>+ "}"
<assignment> ::= <lvalue-expr> "=" <invocation> ";"

<invocation> ::= <identifier> ["<" <type-name> ">"] "(" <argument-list> ")"
<argument-list> ::= <argument> ("," <argument>)*
<argument> ::= <rvalue-expr> | <identifier> "=" <rvalue-expr>

<array-lvalue-expr> ::= "[" [<lvalue-expr> ("," <lvalue-expr>)* ] "]"
<tuple-lvalue-expr> ::= "(" <lvalue-expr> ("," <lvalue-expr>)+ ")" |
                          <lvalue-expr> ("," <lvalue-expr>)+
<lvalue-expr> ::= <identifier> | <array-lvalue-expr> | <tuple-lvalue-expr>
```

# [Basic Rules]

If you need more rule definitions for your parser, you can create additional definitions.

```
<array-rvalue-expr> ::= "[" [<rvalue-expr> ("," <rvalue-expr>)* ] "]"
<tuple-rvalue-expr> ::= "(" <rvalue-expr> ("," <rvalue-expr>)+ ")"
<rvalue-expr> ::= <identifier> | <literal> | <array-rvalue-expr> | <tuple-rvalue-expr>

<literal> ::= <numeric-literal> | <string-literal> | <logical-literal>

<fragment-definition> ::= <fragment-declaration> <body>

<fragment-declaration> ::= "fragment" <identifier> [<generic-declaration>]
                           "(" <parameter-list> ")" "->" "(" <result-list> ")"
<generic-declaration> ::= "<" "?" ["=" <type-name>] ">"
<parameter-list> ::= <parameter> ("," <parameter>)*
<parameter> ::= <identifier> ":" <type-spec> ["=" <literal-expr>]
<result-list> ::= <result> ("," <result>)*
<result> ::= <identifier> ":" <type-spec>

<array-literal-expr> ::= "[" [<literal-expr> ("," <literal-expr>)* ] "]"
<tuple-literal-expr> ::= "(" <literal-expr> ("," <literal-expr>)+ ")"
<literal-expr> ::= <literal> | <array-literal-expr> | <tuple-literal-expr>
```

# [Basic Rules]

If you need more rule definitions for your parser, you can create additional definitions.

```
<type-name> ::= "tensor" | "integer" | "scalar" | "logical" | "string" | "?"
<tensor-type-spec> ::= "tensor" "<" [<type-name>] ">"
<array-type-spec> ::= <type-spec> "[]"
<tuple-type-spec> ::= "(" <type-spec> ("," <type-spec>)+ ")"
<type-spec> ::= <type-name> | <tensor-type-spec> |
                <array-type-spec> | <tuple-type-spec>

<comparison-operator> ::= "<" | "<=" | ">" | ">=" | "==" | "!=" | "in"
<binary-arithmetic-operator> ::= "+" | "-" | "*" | "/" | "^"
<binary-logical-operator> ::= "&&" | "||"
<binary-operator> ::= <comparison-operator>
                    | <binary-arithmetic-operator>
                    | <binary-logical-operator>

<unary-arithmetic-operator> ::= "+" | "-"
<unary-logical-operator> ::= "!"
<unary-operator> ::= <unary-arithmetic-operator>
                   | <unary-logical-operator>
```

# [Basic Rules]

If you need more rule definitions for your parser, you can create additional definitions.

```
<unary-expr> ::= <unary-operator> <rvalue-expr>
<binary-expr> ::= <rvalue-expr> <binary-operator> <rvalue-expr>
<paren-expr> ::= "(" <rvalue-expr> ")"

<if-else-expr> ::= <rvalue-expr> "if" <rvalue-expr> "else" <rvalue-expr>

<loop-iter> ::= <identifier> "in" <rvalue-expr>
<loop-iter-list> ::= <loop-iter> ("," <loop-iter>)*
<comprehension-expr> ::= "[" "for" <loop-iter-list> ["if" <rvalue-expr>]
                         "yield" <rvalue-expr> "]"

<subscript-expr> ::= <rvalue-expr> "[" (<rvalue-expr> |
                     [<rvalue-expr>] ":" [<rvalue-expr>]) "]"

<builtin-name> ::= "shape_of" | "length_of" | "range_of"
                 | "integer" | "scalar" | "logical" | "string"
<builtin-expr> ::= <builtin-name> "(" <rvalue-expr> ")"
```

# [Basic Rules]

If you need more rule definitions for your parser, you can create additional definitions.

```
<rvalue-expr> ::= <identifier>
                | <literal>
                | <binary-expr>
                | <unary-expr>
                | <paren-expr>
                | <array-rvalue-expr>
                | <tuple-rvalue-expr>
                | <subscript-expr>
                | <if-else-expr>
                | <comprehension-expr>
                | <builtin-expr>
                | <invocation>

<assignment> ::= <lvalue-expr> "=" <rvalue-expr> ";"
```

```
<document> ::= <version> <extension>* <fragment-definition>* <graph-definition>
```

# [Basic Rules]

If you need more rule definitions for your parser, you can create additional definitions.

```
<version> ::= "version" <numeric-literal> ";"

<extension> ::= "extension" <identifier>+ ";"
```

# [Examples]

```
fragment conv_layer(
    input: tensor<scalar>,
    channels: integer,
    size: integer[],
    border: string = 'constant',
    padding: (integer,integer)[] = [],
    stride: integer[] = [],
    dilation: integer[] = [],
    groups: integer = 1,
    use_bias: logical = true,
    scope: string )
-> ( output: tensor<scalar> )
{
    planes = shape_of(input)[1] / groups if groups != 0 else 1;
    filter = variable(label = scope + '/filter',
                      shape = [channels, planes] + size);
    bias = variable(label = scope + '/bias', shape = [1, channels])
           if use_bias else constant(shape = [1], value = [0.0]);

    output = conv(input, filter, bias, border = border, padding = padding,
                  stride = stride, dilation = dilation, groups = groups);
}
```

# Input 1

# Output 1

```
def conv_layer(input, channels, size, border, padding, stride, dilation, groups, use_bias, scope):

planes=  tf.shape( input )[ 1 ] /   groups if groups != 0 else 1;
filter= tf.Variable(label=scope+'/filter',shape=[channels,planes]+size);
bias= tf.Variable(label=scope+'/bias',shape=[1,channels]) if use_bias else  tf.constant(shape=[1],value=[0.0]);
output= tf.layers.conv2d(input,filter,bias,border=border,padding=padding,stride=stride,dilation=dilation,groups=groups);

return output
```

# Input 2

```
fragment avg_pool_layer( input: tensor<scalar>, size: integer[],
border: string = 'constant', padding: (integer,integer)[] = [], stride: integer[] = [], dilation: integer[] = [] )
-> ( output: tensor<scalar> )
{
output = avg_pool(input, size = [1,1] + size, border = border,  padding = [(0,0), (0,0)] + padding if length_of(padding) != 0 else [],
stride = [1,1] + stride  if length_of(stride) != 0 else [], dilation = [1,1] + dilation if length_of(dilation) != 0 else []);
}
```

# Output 2

```
def avg_pool_layer(input, size, border, padding, stride, dilation):

output= tf.nn.avg_pool(input,size=[1,1]+size,border=border,padding=[( 0,0) ,( 0,0) ]+ padding if  len( padding ) != 0 else[],
stride=[1,1]+ stride if  len( stride ) != 0 else[],dilation=[1,1]+ dilation if  len( dilation ) != 0 else[]);

return output
```

# [Examples]

```
fragment deconv_layer(
    input: tensor<scalar>,
    channels: integer,
    size: integer[],
    border: string = 'constant',
    padding: (integer,integer)[] = [],
    stride: integer[] = [],
    dilation: integer[] = [],
    output_shape: integer[] = [],
    groups: integer = 1,
    use_bias: logical = true,
    scope: string )
-> ( output: tensor<scalar> )
{
    planes = shape_of(input)[1] / groups if groups != 0 else 1;
    filter = variable(label = scope + '/filter',
                      shape = [channels, planes] + size);
    bias = variable(label = scope + '/bias', shape = [1, channels])
           if use_bias else constant(shape = [1], value = [0.0]);

    output = deconv(input, filter, bias, border = border, padding = padding,
                    stride = stride, dilation = dilation,
                    output_shape = output_shape, groups = groups);
}
```

# Input 3

# Output 3

```
def deconv_layer(input, channels, size, border, padding, stride, dilation, output_shape, groups, use_bias, scope):

planes=  tf.shape( input )[ 1 ] /   groups if groups != 0 else 1;
filter= tf.Variable(label=scope+'/filter',shape=[channels,planes]+size);
bias= tf.Variable(label=scope+'/bias',shape=[1,channels]) if use_bias else  tf.constant(shape=[1],value=[0.0]);
output= tf.layers.conv2d_transpose(input,filter,bias,border=border,padding=padding,stride=stride,dilation=dilation,output_shape=output_shape,groups=groups);

return output
```

# [Examples]

```
fragment simple_recurrent_network(
    x: tensor<scalar>,
    n: integer )
-> ( y: tensor<scalar> )
{
    k = shape_of(x)[0];
    m = shape_of(x)[1];

    W = variable(shape = [n,m], label = 'W');
    U = variable(shape = [n,n], label = 'U');
    b = variable(shape = [1,n], label = 'b');
    s = variable(shape = [k,n], label = 's');

    t = sigmoid(linear(W, x) + linear(U, s) + b);

    y = update(s, t);
}
```

```
def simple_recurrent_network(x, n):

k=  tf.shape( x )[ 0 ];
m=  tf.shape( x )[ 1 ];
W= tf.Variable(shape=[n,m],label='W');
U= tf.Variable(shape=[n,n],label='U');
b= tf.Variable(shape=[1,n],label='b');
s= tf.Variable(shape=[k,n],label='s');
t= tf.sigmoid( linear(W,x)+ linear(U,s)+b);
y= update(s,t);

return y
```

**Input 4  Output  4**

# [Examples]

```
graph ResNet( input ) -> ( output )
{
    input = external(shape = [1,3,224,224]);
    conv1 = resnet_v1_conv_layer(input, channels = 64, size = [7,7],
                                 stride = [2,2], scope = 'conv1');
    pool1 = max_pool_layer(conv1, size = [3,3], stride = [2,2],
                           padding = [(0,1), (0,1)]);
    blocks = resnet_v1_blocks_101(pool1);
    pooled = mean_reduce(blocks, axes = [2,3]);
    logits = resnet_v1_conv_layer(pooled, channels = 1000, size = [1,1],
                                  activation = false, normalization = false,
                                  scope = 'logits');
    output = softmax(logits);
}
```

**Input 5**

# Output 5

```
input= external(shape=[1,3,224,224]);
conv1= resnet_v1_conv_layer(input,channels=64,size=[7,7],stride=[2,2],scope='conv1');
pool1= tf.nn.max_pool(conv1,size=[3,3],stride=[2,2],padding=[( 0,1) ,( 0,1) ]);
blocks= resnet_v1_blocks_101(pool1);
pooled= mean_reduce(blocks,axes=[2,3]);
logits= resnet_v1_conv_layer(pooled,channels=1000,size=[1,1],activation=false,normalization=false,scope='logits');
output= tf.nn.softmax(logits);
```

# [Examples]

```
graph GoogleNet( input ) -> ( output )
{
    input = external(shape = [1, 3, 224, 224]);
    conv1 = conv_layer(input, channels = 64, size = [7,7], stride = [2,2],
                       padding = [(3,2), (3,2)], scope = 'conv1');
    relu1 = relu(conv1);
    pool1 = max_pool_layer(relu1, size = [3,3], stride = [2,2],
                           padding = [(0,1), (0,1)]);
    norm1 = local_response_normalization(pool1, size = [1,5,1,1], alpha = 0.0001,
                                         beta = 0.75, bias = 1.0);
    conv2 = conv_layer(norm1, channels = 64, size = [1,1], scope = 'conv2');
    relu2 = relu(conv2);
    conv3 = conv_layer(relu2, channels = 192, size = [3,3], scope = 'conv3');
    relu3 = relu(conv3);
    norm2 = local_response_normalization(relu3, size = [1,5,1,1], alpha = 0.0001,
                                         beta = 0.75, bias = 1.0);
    pool2 = max_pool_layer(norm2, size = [3,3], stride = [2,2],
                           padding = [(0,1), (0,1)]);
    incept1 = inception(pool2, channels = [64, 96, 128, 16, 32, 32],
                        scope = 'incept1');
    incept2 = inception(incept1, channels = [128, 128, 192, 32, 96, 64],
                        scope = 'incept2');
    pool3 = max_pool_layer(incept2, size = [3,3], stride = [2,2],
                           padding = [(0,1), (0,1)]);
    incept3 = inception(pool3, channels = [192, 96, 208, 16, 48, 64],
                        scope = 'incept3');
```

# Input 6

```
    incept4 = inception(incept3, channels = [160, 112, 224, 24, 64, 64],
                        scope = 'incept4');
    incept5 = inception(incept4, channels = [128, 128, 256, 24, 64, 64],
                        scope = 'incept5');
    incept6 = inception(incept5, channels = [112, 144, 288, 32, 64, 64],
                        scope = 'incept6');
    incept7 = inception(incept6, channels = [256, 160, 320, 32, 128, 128],
                        scope = 'incept7');
    pool4 = max_pool_layer(incept7, size = [3,3], stride = [2,2],
                           padding = [(0,1), (0,1)]);
    incept8 = inception(pool4, channels = [256, 160, 320, 32, 128, 128],
                        scope = 'incept8');
    incept9 = inception(incept8, channels = [384, 192, 384, 48, 128, 128],
                        scope = 'incept9');
    pool5 = avg_pool_layer(incept9, size = [7,7]);
    conv4 = conv_layer(pool5, channels = 1000, size = [1,1], scope = 'conv4');
    logits = softmax(conv4);
}
```

# Output 6

```
input= external(shape=[1,3,224,224]);
conv1= tf.nn.conv2d(input,channels=64,size=[7,7],stride=[2,2],padding=[( 3,2) ,( 3,2) ],scope='conv1');
relu1= tf.nn.relu(conv1);
pool1= tf.nn.max_pool(relu1,size=[3,3],stride=[2,2],padding=[( 0,1) ,( 0,1) ]);
norm1= tf.nn.local_response_normalization(pool1,size=[1,5,1,1],alpha=0.0001,beta=0.75,bias=1.0);
conv2= tf.nn.conv2d(norm1,channels=64,size=[1,1],scope='conv2');
relu2= tf.nn.relu(conv2);
conv3= tf.nn.conv2d(relu2,channels=192,size=[3,3],scope='conv3');
relu3= tf.nn.relu(conv3);
norm2= tf.nn.local_response_normalization(relu3,size=[1,5,1,1],alpha=0.0001,beta=0.75,bias=1.0);
pool2= tf.nn.max_pool(norm2,size=[3,3],stride=[2,2],padding=[( 0,1) ,( 0,1) ]);
incept1= inception(pool2,channels=[64,96,128,16,32,32],scope='incept1');
incept2= inception(incept1,channels=[128,128,192,32,96,64],scope='incept2');
pool3= tf.nn.max_pool(incept2,size=[3,3],stride=[2,2],padding=[( 0,1) ,( 0,1) ]);
incept3= inception(pool3,channels=[192,96,208,16,48,64],scope='incept3');
incept4= inception(incept3,channels=[160,112,224,24,64,64],scope='incept4');
incept5= inception(incept4,channels=[128,128,256,24,64,64],scope='incept5');
incept6= inception(incept5,channels=[112,144,288,32,64,64],scope='incept6');
incept7= inception(incept6,channels=[256,160,320,32,128,128],scope='incept7');
pool4= tf.nn.max_pool(incept7,size=[3,3],stride=[2,2],padding=[( 0,1) ,( 0,1) ]);
incept8= inception(pool4,channels=[256,160,320,32,128,128],scope='incept8');
incept9= inception(incept8,channels=[384,192,384,48,128,128],scope='incept9');
pool5= avg_pool_layer(incept9,size=[7,7]);
conv4= tf.nn.conv2d(pool5,channels=1000,size=[1,1],scope='conv4');
logits= tf.nn.softmax(conv4);
```

1. You need to make compiler which can convert Input# to Output#.
   - Input # is included in Example folder.
2. The basic source code can be used to homework. (You can change it.)
3. The output must be saved as a txt file.
4. The score is classified as below:
   1. 100% : All examples are converted perfectly. (i.e., Input 1 ~ Input 6)
   2. 75% : Five examples which include Input 5 & 6 are converted.
      (i.e., Input 5, Input 6, 3 of Input 1 to 4)
   3. 50% : Four examples which include Input 5 or 6 are converted.
      (i.e., 2 of Input 1 to 4 ,Input 5 or 2 of Input 1 to 4 , Input 6)
   4. 25% : Two examples are converted. (i.e., 2 of Input 1 ~ Input 6)
   5. 0% : otherwise.
5. Upload the Eclipse project(i.e., ~.jj ~.java etc) and Output txt files.
6. Question : asdfwv@naver.com