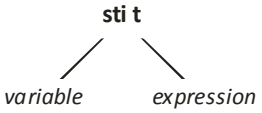
29	*statement*	→	*variable* **:=** *expression*
30	*statement*	→	*procedure_statement*
31	*statement*	→	*compound_statement*
32	*statement*	→	**if** *expression* **then** *statement* **else** *statement*
33	*statement*	→	**while** *expression* **do** *statement*

Note: Function statement must return a list of expressions since compound_statement is a list of expressions.

29 *statement* → *variable* **:=** *expression*



Notes:

1. The goal is to create a single expression whose root is the PCode sti t, where t is the type character representing the type of the expression. The type character t is one of b, c, i, or r representing the type Boolean, character, integer, or real respectively.
2. The sti-PCode is an acronym for "store indirect." By traversing the variable expression first, we put the address in the second to top position on the stack. Then, we traverse the expression putting the value to be stored on top of the stack.
3. The sti-instruction consumes both values on top of the stack storing the value on top of the stack in the address in the next to top position.

*List<Exp***>*** *statement***(***Exp* variable,Exp* expression***)***

1. Issue an error if the *variable* type does not match the *expression* type.
2. Create PCode P
   - 2.1. Label: none
   - 2.2. PCode Operation: store indirect
     sti
   - 2.3. Operand 1: type character
   - 2.4. Operand 2: none

     PCode* P=new PCode("","sti", expression->Type()->TypeChar() ,"");
3. Create Expression E
   - 3.1. Left sub expression: parameter *variable*.
   - 3.2. Right sub expression: parameter *expression*.
   - 3.3. Type: void. assignment statements have no type
   - 3.4. PCode: P created in step 2

     Exp* E=new Exp(variable,expression,ST.TVoid(),P)
4. Create an empty list of expressions.

   List<Exp*>* L=new List<Exp*>;
5. Insert Expression E created in step 3 into the list of expression L.

   L->Insert(E);
6. Return a pointer to the list of the expressions L.

   return L;

30 *statement* → *procedure_statement*

Notes:

1.

*List<Exp***>*** statement***(****Exp* procedure_statement****)***

1. Create an empty list of expressions.
   List<Exp*>* L=new List<Exp*>;
2. Insert parameter procedure_statement, an expression, into the list of expression L.
   L->Insert(procedure_statement);
3. Return a pointer to the list of the expressions L.
   return L;

31 *statement* → *compound_statement*

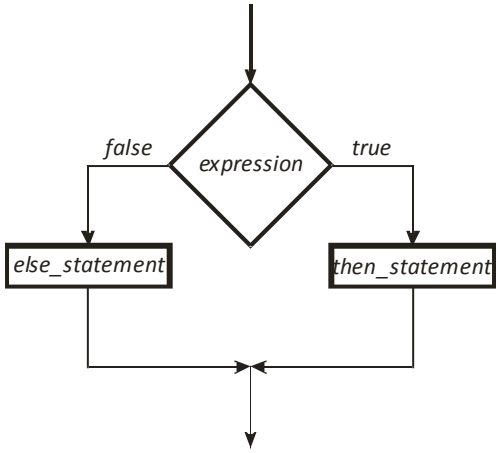
Notes:

1.

*List<Exp***>*** statement***(****List****<****Exp*>* compound_statement****)***

1. Return a pointer to the compound_statement, a list of expressions.
   return compound_statement;

32 *statement* → **if** *expression* **then** *statement* **else** *statement*

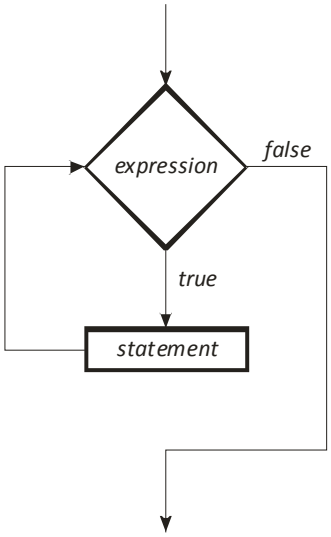


expression
fjp *elsebegin* **//false jump to elsebegin**
then_statement
ujp *elseend* **//unconditionally jump to else end**
elsebegin:
else_statement
elseend:

*List<Exp***>*** statement*
(*Exp* expression*
,List<Exp>*then_statement*
,List<Exp>* else_statement*
)

1. Create the elsebegin label.
   string elsebegin=L.New();
2. Create PCode P
   2.1. Label: none
   2.2. PCode Operator: false-jump
   fjp
   2.3. Operand 1: none
   2.4. Operand 2: elsebegin-label.
   P=new PCode("","fjp","",elsebegin);
3. Create Expression E.
   3.1. Left sub expression: parameter *expression*.
   3.2. Right sub expression: null
   3.3. Type: Void
   3.4. PCode: PCode P created in step 2
   Exp* E=new Exp(expression,0,ST.TVoid(),P);
4. Create a list of expressions, IS, for If Statement.
   List<Exp*>* IS=new List<Exp*>;
5. Insert the fjp-expression in the list of expressions
   IS->Insert(E);
6. Append the list of expressions in parameter *then_stateement*.
   IS->Append(then_statement);
7. Create the PCode label “elsebegin”.
   string elseend=L.New();
8. Create the unconditional-jump to elseend, PCode P.
   8.1. Label: none
   8.2. PCode Operator: unconditional jump
   8.3. Operand 1: none
   8.4. Operand 2: elseend
   P=new PCode("","ujp","",elseend);
9. Create an expression for the ujp-PCode and insert the expression into list IS.
   E=new Exp(ST.TVoid(),P);
   IS->Insert(E);
10. Append the list of expressions in parameter *else_statement*.
    IS->Append(else_statement);
11. Create a PCode and an Expression for the label “elseend”
    P=new PCode(elseend,"","","");
    E=new Exp(ST.TVoid(),P);
12. Insert the “elseend” label onto the list of expressions, IS
    IS->Insert(E);
13. Return a pointer to the list of expressions that contain the If-Statement, IS.
    return IS;

33 *statement* → **while** *expression* **do** *statement*



whilebegin:
expression
fjp *whileend*
statement
ujp *whilebegin*
whileend:

*List<Exp**>>** statement**(**Exp* expression,List<Exp*>* statement**)**

1. Create labels whilebegin and whileend.
   ```
   string whilebegin=L.New();
   string whileend=L.New();
   ```
2. Create a PCode and an Expression for the whilebegin label.
   ```
   P=new PCode(whilebegin,"","","");
   E=new Exp(ST.TVoid(),P);
   ```
3. Create a list of expressions, WS, for the while-statement
   ```
   List<Exp*>* WS=new List<Exp*>;
   ```
4. Insert the whilebegin-expression onto the list WS.
   ```
   WS->Insert(E);
   ```
5. Create the fjp-PCode and Expression.
   ```
   P=new PCode("","fjp","",whileend);
   E=new Exp(expression,0,ST.TVoid(),P);
   ```
6. Insert the fjp-expression onto the list of expressions, WS
   ```
   WS->Insert(E);
   ```
7. Append the list of expressions, representing the loop body, in parameter *statement*.
   ```
   WS->Append(statement);
   ```
8. Create a PCode and an Expression for the unconditional jump to the whilebegin label.
   ```
   P=new PCode("","ujp","",whilebegin);
   E=new Exp(ST.TVoid(),P);
   ```
9. Insert the ujp-expression onto the while statement.
   ```
   WS->Insert(E);
   ```
10. Create a PCode and an expression for the whileend label.
    ```
    P=new PCode(whileend,"","","");
    E=new Exp(ST.TVoid(),P);
    ```
11. Insert the whileend label onto the while statement.
    ```
    WS->Insert(E);
    ```
12. Return a pointer to the list of expressions that contain the While-Statement, WS.
    ```
    return WS;
    ```