36	*procedure_statement*	→	**id**
37	*procedure_statement*	→	**id (** *expression_list* **)**

36 *procedure_statement* → **id**

Exp procedure_statement*(*string* id*)

1. Find parameter *id* in the symbol table.
   Sym S=ST.Find(*id);*
2. **if** parameter id cannot be found in the symbol table **then** issue an error.
   if (!*S*) *yyerror*("**Semantic error - ID cannot be found**");
3. Create an empty list, *L*, of expressions.
   List<Exp>* L*=**new** *List<Exp*>;*
4. **If** symbol *S* is a procedure symbol **then** call function UserSubprogram to create P-Code instructions for the procedure_statement.
   if (*S->IsProcedureSymbol***())return** *UserSubprogram***((***SubprogramSymbol****)***S***,***L***);**
5. **else if** symbol S is a function symbol **then** call function StandardProcedure to create P-Code instructions for the procedure_statement
   else if (*S->IsStandardProcedureSymbol***())**
   return *StandardProcedure***((***StandardProcedureSymbol****)***S***);**
6. **else** issue an error.
   else *yyerror*("**Semantic error - ID must be a procedure**");

37 *procedure_statement* → **id (** *expression_list* **)**

Exp procedure_statement*(*string* id*, *List<Exp*>* L*)

1. Find parameter *id* in the symbol table.
   Sym S=ST.Find(*id);*
2. **if** parameter id cannot be found in the symbol table **then** issue an error.
   if (!*S*) *yyerror*("**Semantic error - ID cannot be found**");
3. **If** symbol *S* is a procedure symbol **then** call function UserSubprogram to create P-Code instructions for the procedure_statement.
   if (*S->IsProcedureSymbol***())return** *UserSubprogram***((***SubprogramSymbol****)***S***,***L***);**
4. **else if** symbol S is a function symbol **then** call function StandardProcedure to create P-Code instructions for the procedure_statement
   else if (*S->IsStandardProcedureSymbol***())**
   return *StandardProcedure***((***StandardProcedureSymbol****)***S***,***L***);**
5. **else** issue an error.
   else *yyerror*("**Semantic error - ID must be a procedure**");

*Exp** *StandardProcedure*(*StandardProcedureSymbol** *S*)
1. Create a new call-standard-procedure P-Code, P.
   1.1. Label: no label
   1.2. PCode Operation: csp
   1.3. Operand 1: no operand
   1.4. Operand 2: CSP identifier found as *S*->*CSPID*()
   *PCode** *P*=**new** *PCode*("","csp","",*S*->*CSPID*());
2. Return a new expression with the P-Code of step 1.
   2.1. Type: void
   2.2. Left subexpression: null
   2.3. Right subexpression null
   2.4. P-Code: PCode created in step 1
   return new Exp(*ST***.***TVoid***(),***P***);**

*Exp** *StandardProcedure*(*StandardProcedureSymbol** *S*,*List*<*Exp**>* *L*)
1. Traverse the list of argument-expressions and create an arg-pcode for each expression.
   *PCode** *P*;
   *Exp** *E*=0;
   for (*L***->***First***();!***L***->***IsEol***();***L***->***Next***()) {**
      *Exp** *A*=*L*->*Member*();
      P=**new** *PCode*("","arg","","");
      E=**new** *Exp*(*E*,*A*,*ST*.*TVoid*(),*P*);
   }
2. Create a call-standard-procedure P-Code and insert it into an expression
   P=**new** *PCode* ("","csp","",*S*->*CSPID*());
   E=**new** *Exp*(*E*,**0**,*ST*.*TVoid*(),*P*);
3. Optionally print the expression.
   E->*PPrint*(*tfs*);
4. Return a pointer to the expression *E*.
   return *E*;

*Exp** *UserSubprogram*(*SubprogramSymbol** *S*,*List*<*Exp**>* *L*)*

1. Create P-Code, P, and expression E for use in this function
   PCode* P;
   Exp* E;
2. Create a mark-stack, mst, P-Code
   2.1. Computer operand 1 of the P-Code. Operand 1 is the difference between the current lexical level and the lexical level of the subprogram
   int *ll*=*ST*.*LexicalLevel*()-*S*->*LexicalLevel*();
   2.2. Create the new P-Code
   P=**new** *PCode*("","mst",*ll*,"");
   2.3. Create an expression for the P-Code
   E=**new** *Exp*(*ST*.*TVoid*(),*P*);
3. Traverse the list of argument-expressions and create an arg-pcode for each expression

```
{  //-------------------------------------------------------------------
   //Obtain the function type FT, and the return type, RT, of the function
   //-------------------------------------------------------------------
   Typ* RT=S->ReturnType();
   PCode* P;
   Exp* E;
   //-------------------------------------------------------------------
   //Put a mark stack, mst, at the bottom of the list.
   //-------------------------------------------------------------------
   int ll=ST.LexicalLevel()-S->LexicalLevel();
   P=new PCode("","mst",ll,"");
   E=new Exp(ST.TVoid(),P);
   //-------------------------------------------------------------------
   //Traverse the list of argument-expressions and create an arg-pcode
   //for each expression
   //-------------------------------------------------------------------
   for (L->First();!L->IsEol();L->Next()) {
     Exp* A=L->Member();
     P=new PCode("","arg","","");
     E=new Exp(E,A,ST.TVoid(),P);
   }
   //-------------------------------------------------------------------
   //Create the cup-pcode and node
   //-------------------------------------------------------------------
   int pc=S->ParameterCount();
   P=new PCode
     (""                  //Label
     ,"cup"                 //P-Code Op - Call User Procedure
     ,pc                  //Operand 1 - Parameter Count
     ,S->ELabel()             //Operand 2 - Entry Label
     );
   E=new Exp(E,0,RT,P);
   E->Print(tfs);
   return E;
```

```
}
```