Question

You will write the recursive descent parser code for the ADALS1 grammar given below. The list of lexeme and token codes are given on page 2.

The code written for the nonterminal symbols <S>, <DECPART>, and <OBJECTDEC>, is already included in the **Answer Document** file below. *You just need to write the code for the rest of the grammar.*

Remember, the nextToken variable is a global variable to all the methods and the method la.getToken() returns the next token of the source code ADALS1 input program. You need to include a call to the error function at each point within your code where a syntax error occurs. Make sure to include the proper syntax error message as a string passed as a parameter to the error function. You do not need to write the code for the error function.

ADALS1 Grammar

```
<S> → procedure ident is <DECPART> begin <SEQOFSTMT> end ; EOI
<DECPART> → <OBJECTDEC> { <OBJECTDEC> }
<OBJECTDEC> → <IDENTLIST> : ( boolean | integer ) ; <IDENTLIST>
→ ident { , <IDENTLIST> }
<SEQOFSTMT> → <STATEMENT> { <STATEMENT> }

<STATEMENT> → null ;
<STATEMENT> → ident := <EXPRESSION> ; <STATEMENT> → if
<CONDITION> then <SEQOFSTMT>

[ else <SEQOFSTMT> ] end if ; <STATEMENT> → while <CONDITION>
loop <SEQOFSTMT> end loop ;

<STATEMENT> → ( get | put ) ( <IDENTLIST> ) ;
<STATEMENT> → newline ;

<CONDITION> → <EXPRESSION>
<EXPRESSION> → <SIMPEXPR> [ ( = | /= | < | <= | > | >= )
<SIMPEXPR> ] <SIMPEXPR> → <TERM> { ( + | - ) <TERM> }
<TERM> → <PRIMARY> { ( * | / | rem ) <PRIMARY> }
<PRIMARY> → ( <EXPRESSION> ) | ident | numlit | true | false
```

ADALS1 Lexeme List

Lexeme	**Token Symbol**
identifier	IDENT
numerical literal	NUMLIT
true	TRUESYM
false	FALSESYM
+	PLUS
not	NOTSYM
–	MINUS
*	TIMES
/	SLASH
rem	REMSYM
:	COLON
=	EQL
/=	NEQ
<	LSS
<=	LEQ
>	GTR
>=	GEQ
(	LPAREN
)	RPAREN
,	COMMA
;	SEMICOLON
end of input	EOI
:=	BECOMES
begin	BEGINSYM
end	ENDSYM
if	IFSYM
then	THENSYM
else	ELSESYM
while	WHILESYM
loop	LOOPSYM
get	GETSYM
put	PUTSYM
newline	NEWLINE
null	NULLSYM
boolean	BOOLSYM
integer	INTSYM
is	ISSYM
procedure	PROCSYM

Answer Document

```
void S( )
{
  nextToken = la.getToken();
  if (nextToken == PROCSYM)
  {
    nextToken = la.getToken();
    if (nextToken == IDENT)
    {
      nextToken = la.getToken();
      if (nextToken == ISSYM)
      {
        nextToken = la.getToken();
        DECPART();
        if (nextToken == BEGINSYM)
        {
          nextToken = la.getToken();
          SEQOFSTMT();
          if (nextToken == ENDSYM)
          {
            nextToken = la.getToken();
            if (nextToken == SEMICOLON)
            {
              nextToken = la.getToken();
              if (nextToken != EOI)
                Print("Program syntactically correct.");
              else
                error("Did not reach the end of the file.");
            }
            else
              error("Missing Semicolon.");
          }
          else
            error("Missing End Symbol.");
        }
        else
          error("Missing Begin Symbol.");
      }
      else
        error("Missing Is Symbol.");
    }
    else
      error("Missing an Identifier.");
  }
  else
    error("Missing Procedure Symbol.");
}
```

```
void DecPart( )
{
  while (nextToken == IDENT) ObjectDec();
}

void ObjectDec()
{
  nextToken = la.getToken();
  while (nextToken == COMMA)
  {
    nextToken = la.getToken();
    if (nextToken = IDENT)
      nextToken = la.getToken();
    else
      error("Missing an Identifier.");
  }

  if (nextToken == COLON)
  {
    nextToken = la.getToken();
    if (nextToken = BOOLSYM || nextToken = INTSYM)
    {
      nextToken = la.getToken();
      if (nextToken = SEMICOLON)
        nextToken = la.getToken();
      else
        error("Missing Semicolon.");
    }
    else
      error("Missing Boolean or Integer Symbol.");
  }
  else
    error("Missing Colon.");
}
```