Caleb Chang 23524938

```
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.");
}




void SeqOfStmt(){
  do{
    Statement();
  }while(nextToken == NULLSYM
```

```
        || nextToken == IDENT
        || nextToken == IFSYM
        || nextToken == WHILESYM
        || nextToken == GETSYM
        || nextToken == PUTSYM
        || nextToken == NEWLINE);
}


void Statement(){
  if(nextToken == NULLSYM){
    nextToken = la.getToken();
    if(nextToken == SEMICOLON){
      nextToken = la.getToken();
    }
    else error("Missing semicolon");
  }
  else if(nextToken == IDENT){
    nextToken = la.getToken();
    if(nextToken == BECOMES){
      nextToken = la.getToken();
      Expression();
      if(nextToken == SEMICOLON){
        nextToken == la.getToken();
      }
      else error("Missing semicolon")
    }
    else error("Missing becomes statement");
  }
  else if(nextToken == IFSYM){
```

```
  nextToken == la.getToken();
  Condition();
  if(nextToken == THENSYM){
    nextToken == la.getToken();
    SeqOfStmt();
    if(nextToken == ELSESYM){
      nextToken == la.getToken();
      SeqOfStmt();
    }
    if(nextToken == ENDSYM){
      nextToken == la.getToken();
      if(nextToken == IFSYM){
        nextToken == la.getToken();
        if(nextToken == SEMICOLON){
          nextToken == la.getToken();
        }
        else error("Missing semicolon");
      }
      else error("Missing if symbol");
    }
    else error("Missing end symbol");
  }
  else error("Missing then symbol");
}
else if(nextToken == WHILESYM){
  nextToken == la.getToken();
  Condition();
  if(nextToken == LOOPSYM){
    nextToken == la.getToken();
    SeqOfStmt();
```

```
    if(nextToken == ENDSYM){
      nextToken == la.getToken();
      if(nextToken == LOOPSYM){
        nextToken == la.getToken();
        if(nextToken == SEMICOLON){
          nextToken == la.getToken();
        }
        else error("Missing semicolon");
      }
      else error("Missing loop symbol");
    }
    else error("Missing end symbol");
  }
  else error("Missing loop symbol");
}
else if(nextToken == GETSYM || nextToken == PUTSYM){
  nextToken == la.getToken();
  if(nextToken == LPAREN){
    do{
      nextToken == la.getToken();
      if(nextToken == IDENT){
        nextToken == la.getToken();
      }
      else error("Missing ident symbol");
    }while(nextToken == COMMA);
    if(nextToken == RPAREN){
      nextToken == la.getToken();
      if(nextToken == SEMICOLON){
        nextToken == la.getToken();
      }
```

```
        else error("Missing semicolon");
      }
      else error( "Missing ) symbol" );
    }
    else error( "Missing ( symbol" );
  }
  else if(nextToken == NEWLINE){
    nextToken == la.getToken();
    if(nextToken == SEMICOLON){
      nextToken == la.getToken();
    }
    else error("Missing semicolon");
  }
  else error ("Missing one of null, ident, if, while, get, put, or
newline);
}



void Expression(){
  SimpExpr();
  if(nextToken == EQL
      || nextToken == NEQ
      || nextToken == LSS
      || nextToken == LEQ
      || nextToken == GTR
      || nextToken == GEQ){
    nextToken = la.getToken();
    SimpExpr();
  }
}
```

```
void SimpExpr(){
  do{
    if(nextToken == PLUS || nextToken == MINUS){
      nextToken = la.getToken();
    }
    Term();
  }while(nextToken == PLUS || nextToken == MINUS);
}


void Term(){
  do{
    if(nextToken == TIMES
          || nextToken == SLASH
          || nextToken == REMSYM){
      nextToken = la.getToken();
    }
    Primary();
  }while(nextToken == TIMES
          || nextToken == SLASH
          || nextToken == REMSYM);
}


void Primary(){
  if(nextToken == LPAREN){
    nextToken == la.getToken()
    Expression();
```

```
    if(nextToken == RPAREN){
      nextToken == la.getToken();
    }
    else error( "Missing ) symbol" );
  }
  else if(nextToken == IDENT
            || nextToken == NUMLIT
            || nextToken == TRUESYM
            || nextToken ==FALSESYM){
    nextToken == la.getToken();
  }
  else error( "Expected (, ident, numlit, true, or false" );
}


void Condition(){
  Expression();
}
```