Lo logra:

El alumno identifica correctamente los elementos estructurales de una gramática de prueba y los problemas relacionados con la gramática de prueba mostrada en la figura 1 por lo que capaz de transformarla a una gramática equivalente susceptible de implementar directamente

programa → *secuencia-sent*
secuencia-sent → *secuencia-sent* **;** *sentencia* | *sentencia*
sentencia → *sent-if* | *sent-repeat* | *sent-assign* | *sent-read* | *sent-write*
sent-if → **if** *exp* **then** *secuencia-sent* **end**
| **if** *exp* **then** *secuencia-sent* **else** *secuencia-sent* **end**
sent-repeat → **repeat** *secuencia-sent* **until** *exp*
sent-assign → **identificador** **:=** *exp*
sent-read → **read identificador**
sent-write → **write** *exp*
exp → *exp-simple op-comparación exp-simple* | *exp-simple*
op-comparación → **<** | **=**
exp-simple → *exp-simple opsuma term* | *term*
addop → **+** | **-**
term → *term opmult factor* | *factor*
opmult → ***** | **/**
factor → **(** *exp* **)** | **número** | **identificador**

Figura 1. Gramática de Prueba

Identifica los símbolos estructurales de una gramática

1. Símbolo inicial. {Programa}

2. Símbolos NO terminales. {Programa, secuencia_sent, sentencia, sent-if, sent-repeat, ...,factor}

3. Símbolos terminales. {if, then, end, else, until, *, /, (,),...,numero, identificador}

4. Reglas de Producción. Las mostradas en la Figura 1.

5. Lenguaje definido por la gramáticas

Identifica los problemas para implementar directamente la gramática mostrada en la figura 1.

1. Recursividad por la izquierda

secuencia-sent → *secuencia-sent* **;** *sentencia* | *sentencia*

exp-simple → *exp-simple opsuma term* | *term*

term → *term opmult factor* | *factor*

2. Para una regla de producción elegida, sus producciones inician con un no terminal

sentencia → *sent-if* | *sent-repeat* | *sent-assign* | *sent-read* | *sent-write*

exp → *exp-simple op-comparación exp-simple* | *exp-simple*

exp-simple → *exp-simple opsuma term* | *term*

term → *term opmult factor* | *factor*

3. Para una regla de producción elegida, sus producciones inician con el(los) mismo(s) símbolos terminales

sent-if → **if** *exp* **then** *secuencia-sent* **end**
| **if** *exp* **then** *secuencia-sent* **else** *secuencia-sent* **end**

exp → *exp-simple op-comparación exp-simple* | *exp-simple*

Proporciona el código

Ejemplo de algunas funciones

a)

```
void sent_if()
{
match("if");
exp();
match("then");
secuencia_sent();
if(tokenActual=="end")
   match("end");
else if(tokenActual=="else")
{
match("else");
secuencia_sent();
match("end");
}
else
   cerr<<"error en la secuencia";


}
```

b)

```
void programa()
{
    //tokenActual Inicia vacio;
tokenActual=obtnSigToken();
//cout<<tokenActual;
//system("pause");
secuencia_sent();

if(tokenActual!="")
cout<<"error";


}
```

c)

```
void sent_assign()
{
if(isalpha(tokenActual[0]))
{
    tokenActual=obtnSigToken();
match(":=");
exp();
}
else
    cerr<<"error";

}
```

Parcialmente lo logra:

Identifica los elementos estructurales de una gramática pero no todas las reglas que presentan una problemática relacionada con la recursividad por la izquierda ni las repetición de símbolos de una gramática que implican la dificultad para decir cual regla de producción se utilizara y proporciona parcialmente el código.

Identifica los símbolos estructurales de una gramática

1. Símbolo inicial. {Programa}

2. Símbolos NO terminales. {Programa, secuencia_sent, sentencia, sent-if, sent-repeat, ...,factor}

3. Símbolos terminales. {if, then, end, else, until, *, /, (,),...,numero, identificador}

4. Reglas de Producción. Las mostradas en la Figura 1.

5. Lenguaje definido por la gramáticas

No Identifica todos problemas para implementar directamente la gramática mostrada en la figura

1. Solo identifica como problema la recursividad por la izquierda.

secuencia-sent → *secuencia-sent* ; *sentencia* | *sentencia*

exp-simple → *exp-simple opsuma term* | *term*

term → *term opmult factor* | *factor*

Proporciona código de las reglas de producción que se pueden implementar directamente.

```
void factor()
{
   if(tokenActual=="(")
      {
         match("(");
         exp();
        // cout<<"**"<<tokenActual;
        // system("pause");
         match(")");

      }

      else if(isdigit(tokenActual[0]))
      {

            match(tokenActual);
      //cout<<"es un numero"<<tokenActual;
       //  system("pause");
      }
      else if(isalpha(tokenActual[0]) || tokenActual[0]=='_')
      {
         match(tokenActual);
      }
      else
      cerr<<"error en el factor";
}
```

No lo logra:

No identifica los elementos estructurales de una gramática libre de contexto ni la problemática relacionada con su implementación por lo que tampoco la proporciona correcta. Confunde los símbolos terminales con los no terminales. eso se refleja en implementaciones como la siguiente.

```
void secuencia_sent()
{

secuencia_sent();
if(token==';')
sentencia()
else
sentencia();
}
```

No se da cuenta que el código proporcionado implica que el programa entre en ciclo al infinito.