Lo logra:

El alumno identifica la necesidad de mejorar el código presentado en la figura 1. Se observa una secuencia de estructuras if-else la cual es dependiente del número de tokens se desean identificar por lo que entre mas tokens se tengan, más grande será la estructura.

```
while(!in.eof())
{
    c=in.get();
    if(c=='(')
    out<<"TokPI"<<endl;
    else if(c==';')
    out<<"TokPyC";
    else if(c=='<')
    out<<"Tokmenor";
    else if(c=='>')
    out<<"Tokmayor";
    else if(isalpha(c)||c=='_')
        {
    else if(c=='/')
        {
 else if(c=='+' || c=='-' || isdigit(c))
 {
else
out<<c;
}
```

Figura 1. Código de un analizador léxico para recuperar el Tokens. Cabe mencionar que el código es más grande que el mostrado. pero se utiliza una herramienta del IDE de programación que permite ocultar la complejidad de la implementación en algunas caso de la estructura.

Identifica la necesidad de usar un arreglo en donde el índice sea un carácter con la finalidad de que al recuperar la letra del archivo de texto que se está analizando, esta sirva también como llave para dirigirse a la función encargada de hacer el posterior análisis y en consecuencia hacer las discriminaciones que correspondan.

Identifica el uso de arreglos asociativos.

Se le guía al alumno para usar arreglos asociativos y en particular el uso de mapeos y punteros a funciones con las siguientes líneas.

```
class cAnalisLexico
{
ifstream in;
ofstream out;
ifstream palabrasReservadas;
list<string> listaPalabras;
vector<cToken*> bufferToken;
typedef void (cAnalisLexico::*punteroFuncionToken)();
char c;

map<char,punteroFuncionToken>  tipoToken;
```

Proporciona parte del código que permite la mejora del código mostrado en la figura 1.

```
while(!in.eof())
{
    c=in.get();

    if(in.eof()|| in.fail())
       break;


    //if(c=='(' || c==';' || c=='
     it=tipoToken.find(c);
     if(it!=tipoToken.end())
     (this->*tipoToken[c])();
    else
    out<<c;
}
```

Parcialmente lo logra:

**Identifica la necesidad de herramientas nuevas que le permitan modelar la solución pero se le dificulta llevarlas a cabo en un lenguaje determinado. Logra avances en cuanto de la estructura de datos map y el uso de punteros a funciones. Incluso asocia las funciones correspondientes a los punteros. Pero se le dificulta dar la parte que lleva a cabo la mejor.**

**Proporciona la estructura de datos map y los punteros a función.**

```
class cAnalisLexico
{
ifstream in;
ofstream out;
ifstream palabrasReservadas;
list<string> listaPalabras;
vector<cToken*> bufferToken;
typedef void (cAnalisLexico::*punteroFuncionToken)();
char c;

map<char,punteroFuncionToken>  tipoToken;
```

**Logra asociar las funciones a cada uno de los punteros**

```
void recuperaTokens()
{
string id;
//tipoToken = map<char,punteroFuncionToken>();
tipoToken['(']=&cAnalisLexico::TokenPI;
tipoToken[';']=&cAnalisLexico::TokenPyC;
tipoToken['<']=&cAnalisLexico::TokenMenor;
tipoToken['>']=&cAnalisLexico::TokenMayor;
tipoToken['[']=&cAnalisLexico::TokenCorIzq;
tipoToken['a']=&cAnalisLexico::TokenID;
tipoToken['_']=tipoToken['a'];
for(int i=97;i<=122;i++)
tipoToken[(char)i]=tipoToken['a'];
for(int i=65;i<=90;i++)
tipoToken[(char)i]=tipoToken['a'];
```

No lo logra:

No tiene clara la idea de arreglos. Tiene problemas al plantear un ciclo y la forma en poner las condiciones a evaluar

```
string num;
num=c;
c=in.get();
if(c='+'){
out<<"(TokIncre,++)";
bufferToken.push_back(new cToken("TokenIncre","++"));
}
else if(c='-'){
out<<"(TokDecre,--)";
bufferToken.push_back(new cToken("TokenDecre","--"));
}
else if(c='='){
out<<"(TokIncre,+=)";
bufferToken.push_back(new cToken("TokenIncre","+="));
```

```
while(c))
    {
        num+=c;
        }
```