Un generador de parser LALR(1) sigue un proceso sistemático para construir un parser a partir de una gramática. A continuación, se describen los pasos que se siguen para generar un parser LALR(1) utilizando la gramática que has proporcionado como base:

**Paso 1: Análisis de la Gramática**

1. **Definición de la Gramática**: La gramática debe estar en una forma adecuada para el análisis. En este caso, la gramática está definida con secciones para **TOKENS**, **KEYWORDS**, y **PRODUCTIONS**.
2. **Identificación de Tokens**: Se definen los tokens que serán utilizados en el análisis, como **number**, **identifier**, **str**, y otros símbolos terminales y no terminales.
3. **Producciones**: Las producciones de la gramática se definen en términos de reglas que describen cómo se pueden formar los diferentes elementos del lenguaje.

**Paso 2: Construcción del Conjunto de Estados**

1. **Construcción del Autómata**: Se construye un autómata que representa los estados del parser. Esto se realiza mediante el uso del algoritmo de construcción de conjuntos de LR (también conocido como el algoritmo de construcción de estados de LR(0)).
2. **Cierre de Conjuntos**: Para cada estado, se realiza un cierre de las producciones que pueden ser alcanzadas a partir de ese estado, lo que incluye las producciones que pueden derivarse a partir de los no terminales.
3. **Transiciones**: Se determinan las transiciones entre estados basándose en los símbolos de entrada. Esto implica observar qué símbolo está en la posición actual de la producción y qué estados se pueden alcanzar al consumir ese símbolo.

**Paso 3: Creación de la Tabla de Parsing**

1. **Construcción de la Tabla LALR**: A partir del autómata, se construye una tabla de parsing LALR(1). Esta tabla incluye:
   * **Acciones**: Que indican si se debe hacer un desplazamiento (shift), una reducción (reduce), aceptar o generar un error.
   * **Gotos**: Que indican a qué estado se debe ir después de una reducción.
2. **Resolución de Conflictos**: En este paso, se resuelven los conflictos de reducción y desplazamiento. Esto se hace aplicando las reglas de precedencia y asociatividad definidas en la gramática.

**Paso 4: Implementación del Parser**

1. **Estructura del Parser**: Se implementa el parser utilizando una estructura de datos como una pila para manejar el estado actual y los símbolos de entrada.
2. **Proceso de Parsing**: El parser lee la entrada token por token y, utilizando la tabla de parsing, decide si debe desplazar, reducir, aceptar o generar un error.
3. **Acciones Semánticas**: Durante el proceso de parsing, se ejecutan las acciones semánticas definidas en las producciones (por ejemplo, **save\_program**, **save\_block**, etc.) para construir la representación del árbol sintáctico o el objeto correspondiente.

**Paso 5: Manejo de Errores**

1. **Detección de Errores**: El parser debe ser capaz de detectar errores de sintaxis. Esto se hace al encontrar un estado en el que no hay una acción válida en la tabla de parsing.
2. **Recuperación de Errores**: Se implementan estrategias para recuperar el parser de errores, como ignorar ciertos tokens o intentar sincronizarse en puntos específicos en la entrada.

**Paso 6: Generación de Código (Opcional)**

1. **Generación de Código Intermedio**: A partir del árbol sintáctico o la representación semántica, se puede proceder a generar código intermedio o código objetivo, dependiendo de los objetivos del compilador.

LALR(1)  
  
Paso 1: Inicialización de la Tabla

Definir la Tabla: Se crea una tabla de parsing que consta de dos partes:

Acciones: Una parte de la tabla que maneja las acciones de desplazamiento (shift), reducción (reduce), aceptación (accept), y error.

Gotos: Otra parte de la tabla que maneja las transiciones entre estados no terminales.

Estructura de la Tabla:

Las filas de la tabla corresponden a los estados del autómata.

Las columnas de la tabla corresponden a los símbolos terminales y no terminales.

Paso 2: Llenar la Tabla de Acciones

Desplazamientos (Shift):

Para cada estado y cada símbolo terminal que puede ser desplazado, se determina la acción de desplazamiento.

Si en un estado S hay una transición a un estado S' al leer un símbolo terminal a, se establece en la tabla:

table[S][a] = shift S'

Reducciones (Reduce):

Para cada estado que tiene una producción completa (es decir, el punto está al final de la producción), se determina si se puede realizar una reducción.

Si el estado S tiene una producción A → α, y el símbolo de look-ahead es b, se establece en la tabla:

table[S][b] = reduce A → α

Es importante verificar que no haya conflictos de reducción (es decir, que no haya múltiples producciones que se puedan reducir en el mismo estado con el mismo símbolo de look-ahead).

Aceptación (Accept):

Si un estado tiene una producción completa que representa el estado de aceptación (por ejemplo, <start> → <program>), se establece en la tabla:

table[S][EOF] = accept

Aquí, EOF representa el final de la entrada.

Errores (Error):

Si no hay ninguna acción válida para un estado y un símbolo de look-ahead, se marca la entrada correspondiente en la tabla como un error.

Paso 3: Llenar la Tabla de Gotos

Transiciones de Goto:

Para cada estado y cada símbolo no terminal, se determina la acción de "goto".

Si en un estado S hay una transición hacia un estado S' al procesar un símbolo no terminal A, se establece en la tabla:

goto[S][A] = S'

Paso 4: Manejo de Conflictos

Conflictos de Desplazamiento y Reducción:

Si hay un conflicto en la tabla (es decir, si hay una entrada que tiene tanto un desplazamiento como una reducción para el mismo estado y símbolo de look-ahead), se debe resolver.

Esto se hace utilizando las reglas de precedencia y asociatividad definidas en la gramática. Por ejemplo, si un operador tiene mayor precedencia que otro, se puede decidir desplazar en lugar de reducir.

Conflictos de Reducción:

Si hay múltiples producciones que pueden ser reducidas en el mismo estado con el mismo símbolo de look-ahead, se debe elegir una de ellas, generalmente la que tiene mayor precedencia o la que aparece primero en la lista de producciones.

Paso 5: Verificación y Optimización

Verificación de la Tabla: Una vez que la tabla está completa, se debe verificar que no haya entradas conflictivas que no se hayan resuelto.

Optimización: En algunos casos, se pueden realizar optimizaciones adicionales para mejorar la eficiencia del parser, como eliminar entradas redundantes o simplificar la tabla.

Paso 6: Finalización

Tabla Completa: Al final de este proceso, se debe tener una tabla de parsing LALR(1) completamente construida que guiará el análisis sintáctico del parser.

Integración con el Parser: La tabla se integra en la implementación del parser, donde se utilizará en combinación con una pila para manejar el estado actual y los símbolos de entrada.

**Paso 1: Definición de los Símbolos de Look-ahead**

Los símbolos de look-ahead son los terminales que pueden aparecer inmediatamente después de una producción en la entrada. El objetivo es identificar qué terminales pueden seguir a cada producción en diferentes contextos.

**Paso 2: Cálculo de los Conjuntos de Primeros (First)**

1. **Definir el Conjunto First**: El conjunto First de un símbolo (terminal o no terminal) es el conjunto de terminales que pueden aparecer como el primer símbolo en una cadena derivada de ese símbolo.
2. **Cálculo del Conjunto First**:
   * Para un símbolo terminal **a**, **First(a) = {a}**.
   * Para un no terminal **A**:
     + Si hay una producción **A → ε** (producción vacía), entonces **First(A)** incluye **ε**.
     + Para cada producción **A → X1 X2 ... Xn**, donde **X1**, **X2**, ..., **Xn** son símbolos:
       - Agregar **First(X1)** a **First(A)**, excluyendo **ε**.
       - Si **X1** puede derivar a **ε**, entonces agregar **First(X2)**, y así sucesivamente.
       - Si todos los **Xi** pueden derivar a **ε**, entonces agregar **ε** a **First(A)**.

**Paso 3: Cálculo de los Conjuntos de Siguientes (Follow)**

1. **Definir el Conjunto Follow**: El conjunto Follow de un no terminal **A** es el conjunto de terminales que pueden aparecer inmediatamente después de **A** en alguna cadena derivada.
2. **Cálculo del Conjunto Follow**:
   * Para el símbolo de inicio de la gramática, se agrega el símbolo de fin de entrada (EOF) al conjunto Follow.
   * Para cada producción de la forma **B → αAβ** (donde **A** es un no terminal y **α** y **β** son secuencias de símbolos):
     + Agregar **First(β)** a **Follow(A)**, excluyendo **ε**.
     + Si **β** puede derivar a **ε** o es vacío, agregar **Follow(B)** a **Follow(A)**.

**Paso 4: Determinación de los Símbolos de Look-ahead para las Producciones**

Una vez que se tienen los conjuntos First y Follow, se pueden determinar los símbolos de look-ahead para cada producción de la siguiente manera:

1. **Para las Producciones que No Terminan en No Terminales**:
   * Si una producción es de la forma **A → α**, los símbolos de look-ahead son simplemente **First(β)**, donde **β** es el símbolo que sigue a **A** en la entrada.
   * Si **β** es vacío (es decir, **A** es el último símbolo de la producción), entonces se utilizan los símbolos de **Follow(A)**.
2. **Para las Producciones que Terminan en No Terminales**:
   * Si una producción es de la forma **A → αBβ**, se consideran los símbolos de **First(β)** para determinar los posibles look-ahead.
   * Si **β** puede derivar a **ε**, entonces se deben considerar los símbolos de **Follow(A)**.
3. **Ejemplo**:
   * Supongamos que tenemos la producción **A → B C**.
   * Para determinar los look-ahead para esta producción:
     + Se calcula **First(C)**. Si **First(C) = {c1, c2}**, entonces los posibles look-ahead son **{c1, c2}**.
     + Si **C** puede derivar a **ε**, entonces se agrega **Follow(A)** a los look-ahead.

**Paso 5: Repetición y Consolidación**

1. **Iterar**: Es posible que necesites repetir los pasos anteriores para asegurarte de que todos los conjuntos First y Follow estén completos y que todos los look-ahead se hayan determinado correctamente.
2. **Consolidación**: Finalmente, se consolidan todos los look-ahead para cada producción, teniendo en cuenta los diferentes contextos en los que pueden aparecer.