UNIVERSIDAD NACIONAL DE SAN AGUSTÍN

FACULTAD DE PRODUCCION Y SERVICIOS

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

Laboratorio - Teoria de la Computación

Gramáticas Independientes de Contexto

Nombre: Apaza Calsin Michael Secarlos
CUI: 20180564

Arequipa - Perú
2021

I OBJETIVOS

- Utilizar bison para crear gramáticas independientes de contexto.
- Manejar los errores al evaluar una gramática.

II EJERCICIOS PROPUESTOS

1. **Ejercicio 1:**
   Crear reglas de producción de gramáticas que solo reconozca "bb" (2 puntos)
   Para este ejercicio, se definió el token "DOBLEB", que almacena 2 veces la letra b, de esta forma solo y exclusivamente se aceptara esta cadena:

```
%%
bb      {return DOBLEB;}
\n      {return NL;}
.       return *yytext;
%%
```

   Para que la cadena sea aceptada tiene que estar seguida por un salto de línea. Se implementó de forma iterativa.

```
%token DOBLEB
%token NL
%%
cadena: DOBLEB NL {printf("Se imprime:\"bb\"\n");};
cadena: cadena DOBLEB NL {printf("Se imprime:\"bb\"\n");};
%%
```

2. **Ejercicio 2:**
   Crear reglas de producción de gramáticas que muestren cadenas pares de letras "a" junto con cadenas pares de letras "b'. (3 puntos)
   Se definieron los tokens A y B de la siguiente forma:

```
%%
a       {return A;}
b       {return B;}
\n      {return NL;}
.       return *yytext;
%%
```

   Luego, al recibirlos en el sintactico2.y pueden formarse cadenas aceptables de la siguiente forma:
   paresA paresB
   Entonces se definió estas nueva reglas de produccion de la siguiente forma:
   paresA: A A paresA | ;
   de forma que se acepte un par de letras A seguidas de paresA, o una cadena vacia, lo mismo con B, obteniendo:

```
%token A
%token B
%token NL
%%
cadena: paresA paresB NL {printf("Se imprime pares de \"a\" y junto a pares de \"b\"\n");};
paresA: A A paresA | ;
paresB: B B paresB | ;
;
cadena: cadena paresA paresB NL {printf("Se imprime pares de \"a\" y junto a pares de \"b\"\n");};
%%
```

3. **Ejercicio 3:**
   Crear reglas de producción de gramáticas que reconozcan igual cantidad de a y también igual cantidad de b. (3 puntos)
   De la misma forma que el ejercicio 2, se definieron los mismos tokens:

```
%%
a       {return A;}
b       {return B;}
\n      {return NL;}
.       return *yytext;
%%
```

   El cambio se dió en sintactico3.y, donde aceptamos a la cadena: S seguida de un salto de línea NL, de forma que S se define de la siguiente forma:
   S: A S B | ;
   De esta forma, cada vez que se use la regla S, entonces si o si tendremos que usar una A y una B, en la misma cantidad(ninguna o varias).

```
%token A
%token B
%token NL
%%
cadena: S NL {printf("Se imprime la misma cantidad de \"a\" y \"b\"\n");};
S: A S B | ;
cadena: cadena S NL {printf("Se imprime la misma cantidad de \"a\" y \"b\"\n");};
%%
int yyerror (char *s)
```

4. **Ejercicio 4:**
   Crear reglas de producción de gramáticas que reconozcan la expresión número + número. (3 puntos)
   Para este ejercicio, definimos 2 Tokens nuevos, el primero NUMERO y el segundo SUMA de la siguiente forma:

```
%%
[0-9]+  {return NUMERO;}
['+']     {return SUMA;}
\n      {return NL;}
.       return *yytext;
%%
```

   Luego, la forma de una cadena aceptable es:
   cadena: NUMERO SUMA NUMERO NL;
   Como lo vemos a continuacion:

```
%token NUMERO
%token SUMA
%token NL
%%
cadena: NUMERO SUMA NUMERO NL {printf("Se imprime una suma de numeros\n");};
cadena: cadena NUMERO SUMA NUMERO NL {printf("Se imprime una suma de numeros\n");};
%%
```

5. **Ejercicio 5:**
   Crear reglas de producción de gramáticas que reconozcan la expresión variable + variable. (3 puntos)
   Aclaración, considerar que la variable solo puede tener letras.
   Para este caso, volvemos a utilizar SUMA, pero añadimos VARIABLE y quitamos NUMERO, esta variable solo puede ser conformada por letras:

```
%%
[a-zA-Z]+   {return VARIABLE;}
['+']       {return SUMA;}
\n          {return NL;}
.           return *yytext;
%%
```

   Entonces la cadena aceptable toma la forma del ejercicio anterior:
   cadena: VARIABLE SUMA VARIABLE
   Como vemos a continuacion:

```
%token VARIABLE
%token SUMA
%token NL
%%
cadena: VARIABLE SUMA VARIABLE NL {printf("Se imprime una suma de variables\n");};
cadena: cadena VARIABLE SUMA VARIABLE NL {printf("Se imprime una suma de variables\n");};
%%
```

6. **Ejercicio 6:**
   Crear reglas de producción de gramáticas que reconozcan la expresión número + variable. (3 puntos)
   Aclaración, considerar que la variable puede tener letras y números. Debe iniciar con letra.
   En este caso, se usaron los tokens NUMERO, SUMA y VARIABLE vistos anteriormente, con cambios pequeños en VARIABLE, ya que ahora puede estar conformada por números después del primer caracter:

```
%%
[a-zA-Z][a-zA-Z0-9]*    {return VARIABLE;}
[0-9]+                  {return NUMERO;}
['+']                   {return SUMA;}
\n                      {return NL;}
.           return *yytext;
%%
```

   De esta forma, la cadena aceptable está conformada por:
   cadena: operacion NL
   Entonces definimos operación de la siguiente forma:
   operacion: VARIABLE SUMA NUMERO | NUMERO SUMA VARIABLE;

Entonces nuestro resultado sería:

```
%token VARIABLE
%token NUMERO
%token SUMA
%token NL
%%
cadena: operacion NL {printf("Se imprime una suma de variable y numero\n");};
operacion: VARIABLE SUMA NUMERO | NUMERO SUMA VARIABLE;
cadena: cadena operacion NL {printf("Se imprime una suma de variable y numero\n");};
%%
```

III CUESTIONARIO

1. **Describir que pasa si no se considera el carácter "." en el analizador léxico (1 punto)**
   Como sabemos, en expresiones regulares el carácter "." identifica a cualquier carácter, y actualmente no afecta considerablemente el procedimiento ya que no operamos con las lecturas, solo identificamos si están correctamente estructuradas.
2. **¿Cuál es el mejor lugar para definir la función main, en el lexer o en el parser? ¿Porque? (2 puntos)**
   Es mejor colocarla en el Parser. Porque el lexer solo identifica los caracteres, mientras que el parser trabaja al nivel de la cadena ya estructurada(con la sintaxis), y generalmente con lo que se busca trabajar es con las cadenas que están correctamente estructuradas.