Practica 2

Analizador Sintáctico LR(1)

Implementación usando Pila de Enteros

Abre el proyecto **analizadorLR1**

Escribe una función para cada uno de los ejemplos, para ejecutar el ejemplo llama la función desde la función **main**.

Ejemplo 1

En este primer ejemplo veras como se utiliza la pila. Escribe una función con nombre "ejemplo1", y ejecútala.

```
Pila pila;

pila.push( 2) ;

pila.push( 3) ;

pila.push( 4) ;

pila.push( 5) ;

pila.muestra();
```

Ejecuta tu programa y observa que el elemento en el tope de la pila es el que se encuentra más a la derecha.

Luego agrega lo siguiente:

```
cout << pila.top() << endl;

cout << pila.top() << endl;
```

Ejecuta de nuevo tu programa. Finalmente, agrega lo siguiente:

```
cout << pila.pop() << endl;
```

```
    cout << pila.pop() << endl;
```

Y ejecuta el programa.

Ejemplo 2

En este ejemplo veras como leer los símbolos de la entrada. Escribe una función con nombre “ejemplo2”, y ejecútala.

```
Lexico lexico("+-+");
    while (! lexico.terminado() ){
        lexico.sigSimbolo();
        cout << lexico.simbolo << endl;
    }
```

Utilizando el código de la practica anterior, modifica el analizador léxico para que reconozca identificadores. Después analiza la entrada “a”.

Ejemplo 3

Gramática 0

Escribe una función “ejemplo3”.

E → <identificador>

	identificador	$	E
0	d2		1
1		**r0 (acept)**	
2		r1	

En una hoja de papel, utiliza la tabla anterior para analiza la cadena: **a**

Ahora, vamos a escribir la tabla en el programa. Para representar la tabla vamos a utilizar un valor entero, los valores del entero representan lo siguiente:

1) Las celdas vacías (sin acción) se representan con el número cero
2) Las reducciones se representan con números negativos, donde -1 representa la reducción 0 (aceptación), -2 es la reducción 1, -3 la reducción 2, -4 que representa la reducción 3, … , -n que representa la reducción n-1.
3) Los desplazamientos y transiciones se representan con los números enteros positivos. El número entero indica el número de desplazamiento o transición respectivamente.

   La tabla anterior quedaría representada como sigue:

	0	1	2
	Id	$	E
0	2	0	1
1	0	-1	0
2	0	-2	0

Para escribir la tabla en el programa crea una variable global como sigue:

```
int tablaLR[3][3]={
          2, 0, 1,
          0, -1, 0,
          0, -2, 0
         };
```

Agrega el siguiente código a tu función. Comenzamos con la inicialización de la pila, y el analizador léxico:

```
    Pila pila;
    int fila, columna, accion;
    bool aceptacion= false;
    Lexico lexico("a");

    pila.push( TipoSimbolo::PESOS );
    pila.push( 0 );
    lexico.sigSimbolo();
```

El primer paso del análisis es encontrar la acción a realizar eso se consigue con:

```
    fila= pila.top();
    columna= lexico.tipo;
    accion= tablaLR[fila][columna];
```

Finalmente mostramos el contenido de la pila, la entrada actual y la acción que se va a realizar.

```
    pila.muestra();

    cout << "entrada: " << lexico.simbolo << endl;

    cout << "accion: " << accion << endl;
```

Dado que el valor de la acción es un número positivo (3) entonces se debe realizar un desplazamiento. Lo que implica meter el símbolo actual a la pila, además del número 3 y pedir un siguiente símbolo.

```
     pila.push( lexico.tipo );

     pila.push( acción );

     léxico.sigSimbolo();
```

Ahora, volvemos a calcular la siguiente acción utilizando el tope de la pila y la entrada, como fila y columna respectivamente.

```
    fila= pila.top();

    columna= lexico.tipo;

    accion= tablaLR[fila][columna];


    pila.muestra();

    cout << "entrada: " << lexico.simbolo << endl;

    cout << "accion: " << accion << endl;
```

Ahora, obtenemos una reducción porque el entero es -2, lo que nos indica que es la reducción 1, como la regla 1 es: E → <id>

Tenemos que sacar 2 elementos de la pila:

```
     pila.pop();

     pila.pop();
```

Luego calculamos la transición utilizando como fila el tope de la pila y como columna el símbolo A (que en este caso es 2).

```
  fila= pila.top();

  columna= 2;  //el no terminal que representa a E
```

```
accion= tablaLR[fila][columna];


//transición

pila.push( 2 ); //el 2 es el entero que representa el no terminal E

pila.push( accion );

pila.muestra();

cout << "entrada: " << lexico.simbolo << endl;

cout << "accion: " << accion << endl << endl;
```

Volvemos, a calcular la siguiente acción y resulta ser -1, lo que implica que es el estado de aceptación.

```
fila= pila.top();

columna= lexico.tipo;

accion= tablaLR[fila][columna];


pila.muestra();

cout << "entrada: " << lexico.simbolo << endl;

cout << "accion: " << accion << endl << endl;

aceptacion= accion == -1;

if (aceptacion) cout << "aceptación" << endl;
```

Nota: En la clase léxico.h en la clase léxico verifica que el valor de identificador sea 0 y el de pesos 1. Los cuales indican las columnas 0 y 1 de la tabla LR, respectivamente.

Ejercicio 1

Gramática: E → <id> + <id>

Realizar el análisis léxico de la cadena: **a+b** Primero en una hoja de papel, luego en el programa.

Tabla LR(1):

	0	1	2	3
	id	+	$	E
0	d2			1
1			**r0 (acept)**	
2		d3		
3	d4			
4			r1	

Nota: En la clase léxico.h en la clase léxico verifica que el valor de identificador sea 0, el del + sea 1, y el de pesos sea 2. Los cuales indican las columnas 0,1 y 2 de la tabla LR, respectivamente.

Ejercicio 2

Gramática: E → <id> + E | <id>

Tabla LR(1):

	0	1	2	3
	id	+	$	E
0	d2			1
1			**r0 (acept)**	
2		d3	r2	
3	d2			4
4			r1	

En este ejercicio la gramática tiene una recursión por lo que habrá que implementar un ciclo (que se termina cuando se acepta la cadena, o hay una entrada invalida). Pero antes de poner el ciclo hay que hacer algunas adecuaciones al programa.

Primero: es necesario distinguir si la acción es un desplazamiento (número positivo) o una reducción (número negativo) y realizar las operaciones necesarias. Por lo que lo primero que necesitaras será una sentencia if parecida a lo siguiente:

```
If ( acción > 0){

//desplazamiento

}

else

if (acción < 0){

//reducción

}

//de lo contrario es un error, porque la celda esta vacía

//else break;            //esta línea se descomenta cuando se agregue el ciclo
```

Otra modificación importante para poder hacer iterativo al algoritmo, es en el caso de las reducciones. Cuando se realiza una reducción se necesita saber el entero que representa el no terminal (ejem. El entero 2 representa al no terminal E). Además, también se necesita conocer cuántos símbolos tiene la regla, para saber cuántos elementos hay que quitar de la pila.

Para resolver estos problemas vamos a utilizar dos arreglos uno para guardar los enteros identificadores de cada no terminal del lado izquierdo de las reglas (idReglas) y otro arreglo que guarde cuantos elementos tiene la regla en su lado derecho (lonReglas).

Los declaramos como variables globales:

```
int idReglas[2]= { 2, 2};

int lonReglas[2]= {3, 0};
```

Estos arreglos tienen dos elementos porque tenemos dos reglas en la gramática:

Regla 1 $E \rightarrow$ <id> + E identificador de E es 2, el total de elementos es 3

Regla 2 $E \rightarrow \varepsilon$ identificador de E es 2, el total de elementos es 0