Analizador Sintáctico LR(1)

Practica 3

Creación de Árboles Sintácticos

En la clase anterior se utilizó una pila de objetos (Estado, No Terminal, Terminal) para realizar el análisis sintáctico. En la presente práctica veremos cómo construir el árbol sintáctico de la gramática que se está analizando.

Gramática 0

<Expresion> ::= <id>

Tabla LR(1)

	id	$	Termino
0	d2		1
1		r0 (acept)	
2		r1	

Escribe el siguiente método

```
void Sintactico::analizaGramatica0(){
   //gramatica0 <Expresion> ::= <id>
   Abre el archivo gramatica0.lr y observa cómo se llenan los arreglos y la tabla
   int tablaLR[3][3]= {  2,   0,      1,
                         0,  -1,      0,
                         0,  -2,      0
```

```
                };

int idReglas[1]= {2};
int lonReglas[1]= {1};
string strReglas[1]= {"Termino"};

int accion, transicion, regla;

pila.push( new Terminal(TipoSimbolo::PESOS) );
pila.push( new Estado( 0 ) );
pila.muestra();
cout << "Entrada: " << lexico.simbolo << "  tipo= " << lexico.tipo << endl;

fila= (pila.top())->id;
columna= lexico.tipo;

accion= tablaLR[ fila ] [columna];
muestraAccion(accion);

//desplazamiento
pila.push( new Terminal( lexico.tipo, lexico.simbolo) );
pila.push( new Estado( accion ) );
lexico.sigSimbolo();

pila.muestra();
cout << "Entrada: " << lexico.simbolo << "  tipo= " << lexico.tipo << endl;
```

```
    fila= (pila.top())->id;

    columna= lexico.tipo;


    accion= tablaLR[ fila ] [columna];

    muestraAccion(accion);


    //reduccion

    regla= -(accion+2); //numero de regla

    for (int i=0; i < lonReglas[regla] ; i++) pila.pop();


    fila= pila.top()->id;

    columna= idReglas[regla]; //el numero de columna donde esta el no Terminal <Expresion>, en
este caso es 2

    transicion= tablaLR[fila][columna]; //transición


    //transicion

    pila.push( new NoTerminal(idReglas[regla], strReglas[regla] ) ); //el 2 es el entero que
representa el no terminal A

    pila.push( new Estado(transicion) );


    pila.muestra();

    cout << "Entrada: " << lexico.simbolo << "  tipo= " << lexico.tipo << endl;

    fila= (pila.top())->id;

    columna= lexico.tipo;

    accion= tablaLR[ fila ] [columna];

    muestraAccion(accion);
```

```
}
```

Modificación para construir árbol sintáctico

```
void Sintactico::analizaGramatica0_1(){
   //gramatica0 <Termino> ::= <id>

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

   int idReglas[1]= {2};
   int lonReglas[1]= {1};
   string strReglas[1]= {"Termino"};

   int accion, transicion, regla;
   NoTerminal *nt;
   Nodo *nodo;

   pila.push( new Terminal(TipoSimbolo::PESOS) );
   pila.push( new Estado( 0 ) );
   pila.muestra();
   cout << "Entrada: " << lexico.simbolo << "  tipo= " << lexico.tipo << endl;

   fila= (pila.top())->id;
   columna= lexico.tipo;

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

```
muestraAccion(accion);

//desplazamiento
pila.push( new Terminal( lexico.tipo, lexico.simbolo) );
pila.push( new Estado( accion ) );
lexico.sigSimbolo();

pila.muestra();
cout << "Entrada: " << lexico.simbolo << "  tipo= " << lexico.tipo << endl;
fila= (pila.top())->id;
columna= lexico.tipo;

accion= tablaLR[ fila ] [columna];
muestraAccion(accion);



nodo= NULL;

//reduccion
regla= -(accion+2); //numero de regla
```

Después de la reducción se crea el nodo del árbol sintáctico

```
switch (regla+1){
  case 1: //Identificador <Termino> ::= <id>
      nodo= new Identificador(&pila);
      break;
```

```
    default:
      for (int i=0; i < lonReglas[regla] * 2 ; i++) pila.pop();
    }


    fila= pila.top()->id;
    columna= idReglas[regla]; //el numero de columna donde esta el no Terminal <Expresion>, en este caso es 2
    transicion= tablaLR[fila][columna]; //transición

    //transición
```

Construcción del nodo, el cual se almacena en el no terminal (nt)

```
    nt= new NoTerminal(idReglas[regla], strReglas[regla] );
    nt->nodo = nodo;

    pila.push( nt ); //el 2 es el entero que representa el no terminal A
    pila.push( new Estado(transicion) );

    pila.muestra();
    cout << "Entrada: " << lexico.simbolo << "  tipo= " << lexico.tipo << endl;
    fila= (pila.top())->id;
    columna= lexico.tipo;
    accion= tablaLR[ fila ] [columna];
    muestraAccion(accion);
```

```
}
```

En el archivo arbolSintactico.h agrega la siguiente clase

```
//clase Identificador
 class Identificador: public Nodo{

 public:
     Identificador(Pila *pila): Nodo() { //constructor de Identificador, primero al constructor de Nodo
        pila->pop();
        this->simbolo= (( Terminal*) pila->pop())->simbolo;
     }

     void muestra(){
       sangria();
       cout << "<ID> " << simbolo << endl;
     }
 };
```

Construye el árbol sintáctico de las siguientes gramáticas

Abre el archivo gramatica#.lr correspondiente al ejercicio, para que llenes los arreglos y la tabla

No olvides modificar la clase TipoSimbolo que está en el archivo Lexico.h, en cada gramática

Gramática 1

1) <Termino> ::= id
2) <Termino> ::= entero

	id	entero	$	Termino
0	d2	d3		1
1			r0	
2			r1	
3			r2	

Código de la reducción

```
//reduccion
regla= -(accion+2); //numero de regla
switch (regla+1){
 case 1: //Identificador
    nodo= new Identificador(&pila);
    break;
 case 2: //Entero
    nodo= new Entero(&pila);
    break;
 default:
  for (int i=0; i < lonReglas[regla] * 2 ; i++) pila.pop();
}
```

Gramática 2

1) <Expresion> ::= <Termino>
2) <Expresion> ::= <Termino> + <Termino>
3) <Termino> ::= id
4) <Termino> ::= entero

	id	entero	+	$	Expresion	Termino
0	d3	d4			1	2
1				r0 (acept)		
2			d5	r1		
3			r3	r3		
4			r4	r4		
5	d3	d4				6
6				r2		

```
//reduccion
regla= -(accion+2); //numero de regla

switch (regla+1){
  case 1: //<Expresion> ::= <Termino>
      pila.pop();
      nodo= pila.pop()->nodo;
      break;

  case 2: //<Expresion> ::= <Termino> + <Termino>
//        nodo= new Suma(&pila);
      break;
```

```
  case 3: //Identificador
      nodo= new Identificador(&pila);
      break;

  case 4: //Entero
      nodo= new Entero(&pila);
      break;

  default:
    for (int i=0; i < lonReglas[regla] * 2 ; i++) pila.pop();
}
```

Gramática 3

1) <Expresion> ::= <Expresion> opMul <Expresion>
2) <Expresion> ::= <Expresion> opSuma <Expresion>
3) <Expresion> ::= <Termino>
4) <Termino> ::= id
5) <Termino> ::= entero

	id	entero	opMul	opSuma	$	Expresion	Termino
0	d3	d4				1	2
1			d5	d6	r0 (acept)		
2			r3	r3	r3		
3			r4	r4	r4		
4			r5	r5	r5		
5	d3	d4				7	2
6	d3	d4				8	2
7			r1	r1	r1		
8			d5	r2	r2		

En las gramáticas de expresiones es recomendable definir una clase Expresión de donde heredan los otros objetos, la cual tienen hijos izquierdo y derecho.

```
class Expresion: public Nodo{
   protected:
    Expresion *izq, *der;
  public:
     Expresion(){
        izq= der= NULL;
        sig= NULL;
     }
};
```