SINTAXIS Y SEMÁNTICA DE LOS LENGUAJES

CURSO K2006

GRUPO 9

TP N°2 GRUPAL

INTEGRANTES	LEGAJO
Roberto Rodas	2081301
Alex Fiorenza	2089865
Oscar Mercado	2083012
Joaquín Fatur	2079847

TP Sintaxis y Semanticas de los Lenguajes

Estructura del proyecto ⚙

```
.
├── README.md
|-- imgs -> Recursos didácticos para la documentación
├── libs
|   ├── automata.c -> Funcionalidades dedicadas a la manipulacion de automatas
|   ├── interface.c -> Funcionalidades dedicadas a la interfaz de usuario
|   ├── utils.c -> Funcionalidades varias
|──main.c
```

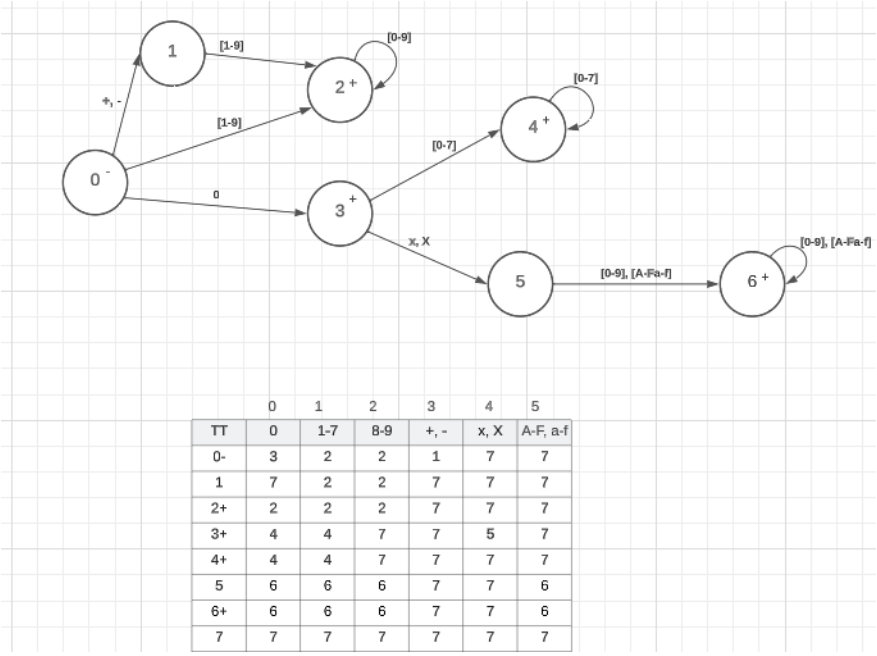
Inicializar el proyecto

Consideraciones

- En caso de correr a traves de un archivo este debe ser creado en la carpeta root con el nombre **expresionAritmetica.txt**
- Los automatas desarrollados consumen toda la cadena y al final se determina si terminaron en un estado de rechazo

Documentación

Decisiones acerca del punto 1

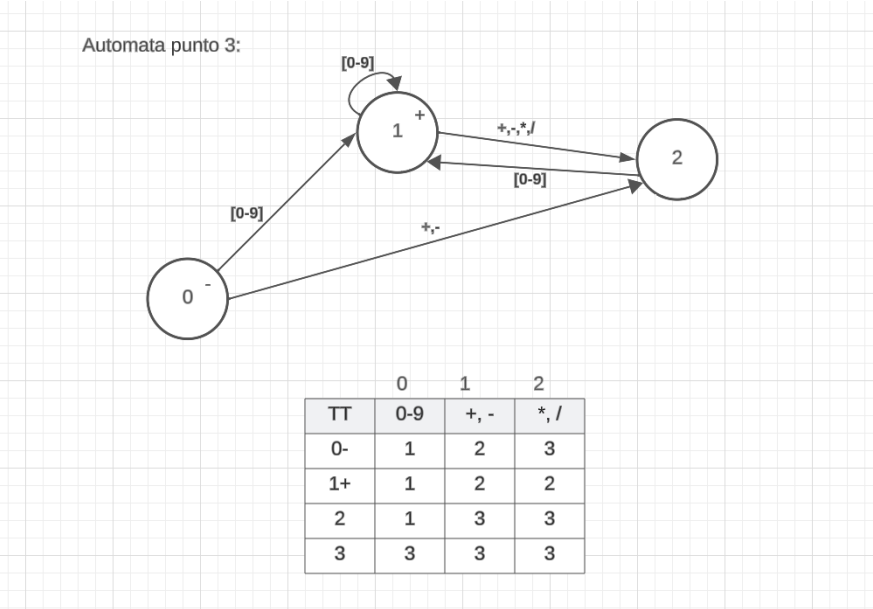


	0	1	2	3	4	5
TT	0	1-7	8-9	+, -	x, X	A-F, a-f
0-	3	2	2	1	7	7
1	7	2	2	7	7	7
2+	2	2	2	7	7	7
3+	4	4	7	7	5	7
4+	4	4	7	7	7	7
5	6	6	6	7	7	6
6+	6	6	6	7	7	6
7	7	7	7	7	7	7

- Se decidio utilizar un unico automata para la detección de tokens. En un principio se consideraron 3 automatas (uno para cada sistema) y luego se decidio unificarlos en uno solo. Al momento de la implementacion se considero mucho más simplificado el automata unificado y de hecho para el codigo termino siendo mucho mas corto.
- En la creación del automata no se tuvo en cuenta el caracter "$" ya que si bien la consigna lo pide, se decidio realizar la separacion entre caracteres desde el codigo y no desde el automata. No se espera que las cadenas sean demasiado largas para encontrar un beneficio en comprobar caracter a caracter si ya cayó en un estado de error

```
    while (c != '\0')
    {
        if (c != '$') //Se considera el signo $ como un caracter  de separación
        {
            estado = tt[estado][columnaNumeros(c)];
        }
        else
        {
            incrementarContadoresSegunEl(estado, &decimales, &octales, &hexadecimales, &noReconocidos);
            estado = 0;
        }
        c = cadena[++i];
    }
```

Decisiones acerca del punto 3



	0	1	2
TT	0-9	+, -	*, /
0-	1	2	3
1+	1	2	2
2	1	3	3
3	3	3	3

- Para resolver el desafio de `resolverOperacion` se utilizo la siguiente metodologia:

1. Se separa la cadena de toda la operación en 2 arrays. Uno de operandos y otro de operadores. Sino comienza con signo, se le agrega un + a los operadores

```
 char operadores[50];
int operandos[50];
int i = 0, j = 0, k = 0;
char enteroCadena[12];
int terminos[50];

// La funcion devuelve 1 si es + o -. Por ejemplo la cadena +3-4 devuelve 1 pero 3-4 devuelve 0 pues comienza con un numero
if (esSumaOResta(expresionAritmetica[0])) // para que comience siempre con un signo
{
    operadores[0] = expresionAritmetica[0];
    i++; // el siguiente bucle va a comenzar en la posición 1 del array
}
else
{
    operadores[0] = '+';
}
```

2. Luego se procede a cargar los operandos en el array de operandos y los operadores en el array de operadores.

```
 for (; expresionAritmetica[i]; i++) // cargar operadores y operandos
{
    if (isdigit(expresionAritmetica[i]))
    {
        enteroCadena[j++] = expresionAritmetica[i];
    }
    else
    {
        enteroCadena[j] = '\0';
        j = 0;
        operandos[k++] = cadenaADecimal(enteroCadena);

        operadores[k] = expresionAritmetica[i];
    }
}
enteroCadena[j] = '\0';
operandos[k] = cadenaADecimal(enteroCadena);

const int longitud = k;
```

3. Se cargan los terminos operando los y /, con ayuda de la variable primerOperando para guardar el resultado parcial de ir aplicando o/, los + y - sirven para saber donde finaliza un termino y guardar el resultado del termino en el array terminos.

```
// resolver los terminos y cargarlos en el array terminos
  int primerOperando = operandos[0];
  j = 0;
  for (i = 1; i <= longitud; i++)
  {
      if (esSumaOResta(operadores[i]))
      {
          terminos[j++] = primerOperando;
          primerOperando = operandos[i];
      }
      else
      {
          primerOperando = calcular(primerOperando, operandos[i], operadores[i]);
      }
  }
  terminos[j] = primerOperando;
```

4.Para calcular el resultado final, arranca en 0 y va restando o sumando los terminos segun el operador que le corresponda. Se utiliza la funcion `calcular` para realizar las operaciones.

```
   // operar los terminos
      int resultado = 0;
      j = 0;
      for (i = 0; i <= longitud; i++)
      {
          if (esSumaOResta(operadores[i]))
          {
              resultado = calcular(resultado, terminos[j++], operadores[i]);
          }
      }
```

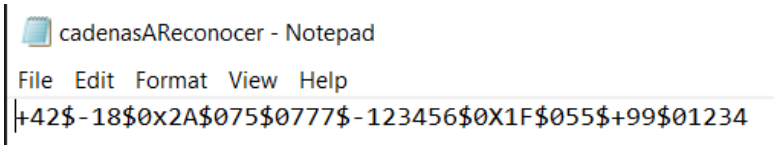
PRUEBAS DEL PUNTO 1

Probemos con la siguiente cadena la cual contiene 4 números decimales, 4 octales y 2 hexadecimales: +42$-18$0x2A$075$0777$-123456$0X1F055+99$012

Ingresando la cadena por consola:

```
Ingrese:
0. Leer el archivo cadenasAReconocer.txt
1. Escribir los numeros separandolos por $
1
Escriba los numeros separandolos por $ +42$-18$0x2A$075$0777$-123456$0X1F$055$+99$01234
Decimales:      4
Octales:        4
Hexadecimales:  2
No reconocidos: 0
```

Leyéndolo desde el archivo de texto cadenasAReconocer.txt:




```
Ingrese:
0. Leer el archivo cadenasAReconocer.txt
1. Escribir los numeros separandolos por $
0
Decimales:      4
Octales:        4
Hexadecimales:  2
No reconocidos: 0
```

Modificando la cadena, añadiendo al final, por ejemplo, un * provoca que la cadena no cumpla con el alfabeto, por lo que no es una cadena válida para analizar:

+42$-18$0x2A$075$0777$-123456$0X1F055+99$012*

```
Ingrese:
0. Leer el archivo cadenasAReconocer.txt
1. Escribir los numeros separandolos por $
1
Escriba los numeros separandolos por $ +42$-18$0x2A$075$0777$-123456$0X1F$055$+99$012*
No verifica el alfabeto la cadena: +42$-18$0x2A$075$0777$-123456$0X1F$055$+99$012*
```

Si la modificamos añadiendo al final un – en vez de un *, la cadena si verificará el alfabeto, pero ese último número no será reconocido por el autómata:

+42$-18$0x2A$075$0777$-123456$0X1F055+99$012-

```
Ingrese:
0. Leer el archivo cadenasAReconocer.txt
1. Escribir los numeros separandolos por $
1
Escriba los numeros separandolos por $ +42$-18$0x2A$075$0777$-123456$0X1F$055$+99$012-
Decimales:      4
Octales:        3
Hexadecimales:  2
No reconocidos: 1
```

PRUEBAS DEL PUNTO 3

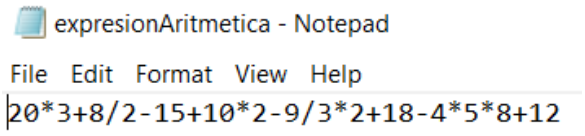
Probemos resolver la siguiente expresión aritmética:

20*3+8/2-15+10*2-9/3*2+18-4*5*8+12

Ingresando la cadena por consola:

```
Ingrese:
0. Leer el archivo expresionAritmetica.txt
1. Escribir la operacion aritmetica simple
1
Escriba una operacion aritmetica simple (puede tener multiples +, -, *, /):  20*3+8/2-15+10*2-9/3*2+18-4*5*8+12
El resultado es: -67
```

Leyéndolo desde el archivo expresionAritmetica.txt:




```
Ingrese:
0. Leer el archivo expresionAritmetica.txt
1. Escribir la operacion aritmetica simple
0
El resultado es: -67
```

Nuevamente, si añadimos, por ejemplo, una “a” al principio de la expresión, la cadena será rechazada al no verificar el alfabeto:

a20*3+8/2-15+10*2-9/3*2+18-4*5*8+12

```
Ingrese:
0. Leer el archivo expresionAritmetica.txt
1. Escribir la operacion aritmetica simple
1
Escriba una operacion aritmetica simple (puede tener multiples +, -, *, /):  a20*3+8/2-15+10*2-9/3*2+18-4*5*8+12
La operacion tiene caracteres no validos
```

Si, por ejemplo, en alguna parte de la expresión aparece un “++”, la cadena verificará el alfabeto, pero no será reconocida por el autómata, por lo que la resolución de la expresión no se llevará a cabo al ser una expresión aritmética invalida:

20*3++8/2-15+10*2-9/3*2+18-4*5*8+12

```
Ingrese:
0. Leer el archivo expresionAritmetica.txt
1. Escribir la operacion aritmetica simple
1
Escriba una operacion aritmetica simple (puede tener multiples +, -, *, /):  20*3++8/2-15+10*2-9/3*2+18-4*5*8+12
Operacion no valida
```