



Universidad Tecnológica Nacional - Facultad Regional Buenos Aires

Sintaxis y Semántica de los Lenguajes

2022

Trabajo Práctico Grupal N° 2

Tema: AUTOMATAS

Curso: K2006

Profesora: Roxana Leituz

Fecha Estipulada de Entrega: 18 de Septiembre

Grupo N° 5

Alumno	Legajo
Beretta Chiara Sofía	204.619-2
Gribnicow Irina Pérez	178.165-0
Iglesias Zoe	203.734-8
Pangaro Lucas	164.142-6
Romano Mateo Agustín	204.018-9
Santucho Gianluca	204.050-5

	Fecha	Nota	Observaciones	Fecha Devol.	Firma Docente
Entrega	18/09				
Corrección					

Ejercicio 1: Dada una cadena que contenga varios números que pueden ser decimales, octales o hexadecimales, con o sin signo para el caso de los decimales, separados por el carácter '&', reconocer los tres grupos de constantes enteras, indicando si hubo un error léxico, en caso de ser correcto contar la cantidad de cada grupo. Debe diagramar y entregar el o los autómatas utilizados y las matrices de transición. La cadena debe ingresar por línea de comando o por archivo.

Para desarrollar este ejercicio antes tenemos que saber qué forma tienen los datos mencionados.

Las constantes hexadecimales comienzan con cero seguidas de una equis (mayúsculo o minúscula). Posteriormente puede continuar con un número (del cero al nueve) o una letra (de la 'a' a la 'f') tanto mayúscula como minúscula.

Algunos ejemplos válidos para este tipo de constante son: 0X123, 0x123, 0xAb, 0X0, 0Xa9f.

Las constantes octales tienen la particularidad que comienzan siempre con 0 y luego tienen un número del cero al siete.

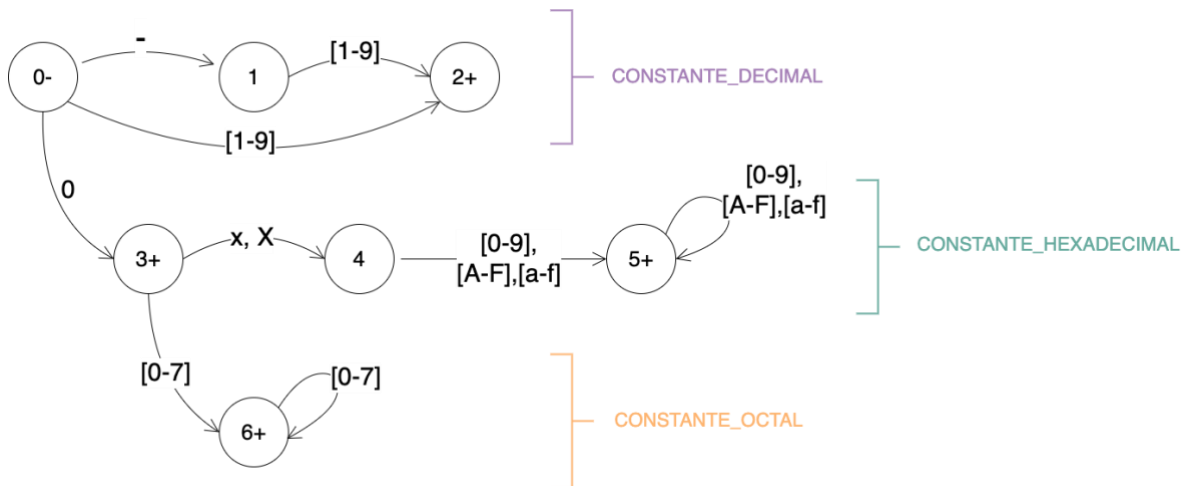
Algunos ejemplos son: 012, 0034, 076.

Como mención especial consideramos que el 0 pertenece a las constantes octales, aunque también podría incluirse dentro de las decimales. El único motivo de esto es simplificar el código, cuando el usuario ingresa el 0 podemos decir que el número será un octal y quedaron en ese estado preguntando por los siguientes, salvo que continúe con 'x', en ese caso será un hexadecimal (siempre y cuando el resto de caracteres también cumplan la condición).

Finalmente, las constantes decimales son todos los números que se pueden formar con los caracteres del cero al nueve, también incluye los negativos, es decir los que comienzan con '-'. Pero no reconoce los que tienen como primer carácter numérico un 0. Es decir 03, -03.

Algunos ejemplos de caracteres decimales válidos son: 123, -45, 7, -9, 10.

En representación de todo lo anterior se realizó el siguiente autómata con su respectiva tabla.



	-	0	1	2	3	4	5	6	7	8	9	a, A	b, B	c, C	d, D	e, E	f, F	x, X
0-	1	3	2	2	2	2	2	2	2	2	2	7	7	7	7	7	7	7
1	7	2	2	2	2	2	2	2	2	2	2	7	7	7	7	7	7	7
2+	7	2	2	2	2	2	2	2	2	2	2	7	7	7	7	7	7	7
3+	7	6	6	6	6	6	6	6	6	7	7	7	7	7	7	7	4	4
4	7	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	7	7
5+	7	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	7	7
6+	7	6	6	6	6	6	6	6	6	7	7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7

Algunos detalles sobre el código...

Se comenzó realizando la declaración de nombres de constantes con el tipo de dato `enum`, esto nos permite una mayor expresividad en el código. (Estos nombres se usarán más adelante en un `switch`).

La función `main` es muy sencilla y se utiliza para llamar al resto de funciones que harán el trabajo duro del programa; sin embargo, es en este momento que el usuario elige como ingresar la cadena deseada. Puede optar por un ingreso en la línea de comandos (realizado con un `scanf`) o bien mediante un archivo, para esta alternativa además se le pedirá el nombre del archivo y con este se llama a una función.

```

36     printf("Escriba una opcion: ");
37     scanf ("%d", &menu);
38     system("clear");
39
40     switch (menu){
41     case 1 :
42         printf("Ingrese la cadena de caracteres numericos (Hexadecimales, Octales o Decimales): ");
43         scanf ("%s", vector_palabras);
44         break;
45     case 2 :
46         printf("Ingrese el nombre del archivo (ej: entrada.txt): ");
47         scanf ("%s", archivo);
48         ingreso_por_archivo(archivo, vector_palabras);
49         break;
50     default:
51         printf ("No ingreso una opcion valida \n");
52     }

```

Cualquiera sea la opción elegida se guarda la cadena en el `vector_palabras` que luego es enviado a la función `separar_y_reconocer`.

```

54     separar_y_reconocer(vector_palabras, f_salida); // PASO EL ARRAY CON LAS PALABRAS A RECONOCER
55     printf("Archivo creado exitosamente. Para visializarlo escriba: 'cat salida.txt'\n");
56

```

Dentro de esta función se crean variables del tipo `int` que serán pasadas por referencia a una nueva función `reconocer`, de esta forma `reconocer` puede “devolver” la cantidad de constataste de cada tipo.

Pero antes de ello se tiene que obtener la sub-cadena de `char` que representa cada número. Como dice el enunciado cada una de estas sub-cadenas se encuentra separa por el carácter '&'; por lo tanto, con la utilización de la función `strtok` y pasándole como argumento el puntero al vector y el centinela (&) se obtiene cada cadena a analizar.

```

91     str = strtok(vector, sentinela); // str
92     while (str != NULL){ // usa
93         reconocer(str, f_salida, &cant_oct, &cant_deci, &cant_hexa);
94         str = strtok(NULL, sentinela);
95     }

```

La función `reconocer` es la más compleja de todas, es la encargada de recorrer cada sub-cadena y decir a que tipo de constante pertenece.

Para esto comienza con un `while(1)` que se utiliza para que cicle indefinidamente. Dentro de este ciclo tenemos un `switch` que utiliza como estados los nombres definidos al principio con ayuda de `enum`.

En cada uno de estos estados procura reconocer un carácter y dependiendo de su contenido cambia el estado.

Para comenzar tenemos el estado inicio en el que se lee el primer carácter de la sub-cadena. Si este carácter es un 0 es posible que el número leído sea un octal o un hexadecimal. Es por esto que el estado cambia a OCTAL_O_HEXA.

```
case INICIO:
    if(valor[i] == '0')
        estado = OCTAL_O_HEXA;
    else if(isdigit(valor[i]))
        estado = CONSTANTE_DECIMAL;
    else if (valor[i] == '-') {
        estado = DECIMAL_NEGATIVO;
    }
    else
        estado = DESCONOCIDO;
    i++;
    break;
```

Si el carácter que se reconoció no fue un '0' esta la posibilidad que sea un decimal asique con la función `isdigit` se pregunta si dicho carácter es un número, si lo es cambia el estado a CONSTATNE_DECIMAL.

(`isdigit` también reconoce al '0' como un número, por lo que si el carácter fuese '0' también se validaría esta condición. Sin embargo, como los programas en C corren línea por línea al cumplirse la condición anterior ya no entraría en esta).

Finalmente tenemos dos opciones más, si el carácter leído no es un número, puede ser que sea un menos '-', carácter valido para los decimales, por lo que se envía a un estado llamado DECIMAL_NEGATIVO. Y si no es ni un numero ni '-' el estado cambia a DESCONOCIDO.

Si el estado llega a desconocido se imprime que la cadena no fue reconocida y termina el programa

```
case DESCONOCIDO:
    fprintf(f_salida, "%-20s\tNO RECONOCIDA\n", valor);
    return;
```

Continuando en orden y recordando si el carácter era un '0' se enviaba al estado OCTAL_O_HEXA.

Es en este estado en que se pregunta por el carácter siguiente. Si es '\0' quiere decir que la cadena termino y el numero era 0 (octal), se guarda en un archivo y retorna rompiendo el ciclo while.

Si no es '\0' seguimos preguntando si es una equis mayúscula o minúscula, si lo es todo indica que el número es un hexadecimal por lo que el estado cambia a CONSTANTE_HEXADECIMAL. En cambio, si lo que se lee es un numero entre el 0 y el 7 es un octal (CONSTANTE_OCTAL).

```

case OCTAL_0_HEX:
    if(valor[i] == '\\0'){
        (*cant_oct)++;
        fprintf(f_salida, "%-20s\\t %d OCTAL\\n", valor, *cant_oct);
        return;
    }
    else if(valor[i] == 'x' || valor[i] == 'X')
        estado = CONSTANTE_HEXADECIMAL2;
    else if(isdigit(valor[i]) && valor[i] - '0' <= 7)
        estado = CONSTANTE_OCTAL;
    else
        estado = DESCONOCIDO;
    i++;
    break;

```

Otro estado especial es el DECIMAL_NEGATIVO a este entramos cuando el primer carácter de la sub-cadena era un '-', ahora preguntamos si el carácter que le sigue es un dígito salvo el 0 (porque no tendría sentido tener -0) si es un dígito, el estado cambia a CONSTANTE_DECIMAL

```

case DECIMAL_NEGATIVO: //signo menos
    if(valor[i] == '\\0'){ // ¿FIN?
        estado = DESCONOCIDO;
    }
    else if(isdigit(valor[i]) && valor[i] - '0' > 0)
        estado = CONSTANTE_DECIMAL;
    else
        estado = DESCONOCIDO;
    i++;
    break;

```

Para los siguientes estados es siempre lo mismo, CONSTANTE_OCTAL, CONSTANTE_DECIMAL y CONSTANTE_HEXADECIMAL, realizan tres preguntas. Si estoy en ese estado y **llega el carácter '\\0'** quiere decir que la sub-cadena terminó, por lo que se guarda en un archivo de salida el valor de la constante, el tipo y su cantidad, retornando y saliendo del `while(1)`.

Si el carácter leído no es '\\0' **pregunto si se sigue cumpliendo la condición que sea octal, decimal o Hexa**. Si la cumple, el estado sigue siendo el mismo y en la próxima vuelta del ciclo pregunto por el carácter siguiente.

Tercero y por último si no cumple con ninguna de las anteriores quiere decir que la cadena NO ES RECONOCIDA, el estado cambia a DESCONOCIDO.

```

case CONSTANTE_HEXADECIMAL2:
    if(valor[i] == '\\0'){
        (*cant_hexa)++;
        fprintf(f_salida, "%-20s\\t %d HEXADECIMAL\\n", valor, *cant_hexa);
        return;
    }
    else if(isxdigit(valor[i]))
        estado = CONSTANTE_HEXADECIMAL2;
    else
        estado = DESCONOCIDO;
    i++;
    break;

```

```

case CONSTANTE_DECIMAL:
    if(valor[i] == '\\0'){
        (*cant_deci)++;
        fprintf(f_salida, "%-20s\\t %d DECIMAL\\n", valor, *cant_deci);
        return;
    }
    else if(isdigit(valor[i]))
        estado = CONSTANTE_DECIMAL;
    else
        estado = DESCONOCIDO;
    i++;
    break;

```

```

case CONSTANTE_OCTAL:
    if(valor[i] == '\\0'){
        (*cant_oct)++;
        fprintf(f_salida, "%-20s\\t %d OCTAL\\n", valor, *cant_oct);
        return;
    }
    else if(isdigit(valor[i]) && valor[i] - '\\0' <= 7)
        estado = CONSTANTE_OCTAL;
    else
        estado = DESCONOCIDO;
    i++;
    break;

```

Por último, podemos mencionar un estado extra de los Hexadecimales, a este estado entra luego de haber leído 0x y se asegura que posteriormente de ese inicio de cadena continúe con por lo menos otro valor. De esta manera no permite las cadenas "0x" o "0X" como hexadecimal.

```

case CONSTANTE_HEXADECIMAL1:    //Se asegura
    if(isxdigit(valor[i]))
        estado = CONSTANTE_HEXADECIMAL2;
    else
        estado = DESCONOCIDO;
    i++;
    break;

```

Como se puede ver, para la resolución de este problema se tomó la decisión de cambiar el estado en cada uno de los caracteres y que se retorne inmediatamente el error, esto permite que si tengo una cadena que contine un error al principio no continúe procesándola. Por ejemplo, la cadena "0x^1234ab4578f" deja de ser procesada en el tercer carácter ya que entra al estado DESCONOCIDO y allí se hace un `return` que rompe el `while(1)`.

Ejemplo del programa corriendo con ingreso desde archivo

```
Entrada.txt
0xFF&127&0159&0xaBb1&0Xx&0&010&09&127ABACB&07&-10

Ejercicio_1 -- zsh -- 80x24
Ingrese el nombre del archivo (ej: entrada.txt): entrada.txt
Archivo creado exitosamente. Para visualizarlo escriba: 'cat salida.txt'
lpangaro@MacBook-Air-de-Lucas Ejercicio_1 % cat salida.txt
Numero          Cant    Tipo
0xFF             1    HEXADECIMAL
127              1    DECIMAL
0159             NO RECONOCIDA
0xaBb1           2    HEXADECIMAL
0Xx              NO RECONOCIDA
0                1    OCTAL
010              2    OCTAL
09               NO RECONOCIDA
127ABACB         NO RECONOCIDA
07               3    OCTAL
-10              2    DECIMAL
lpangaro@MacBook-Air-de-Lucas Ejercicio_1 %
```

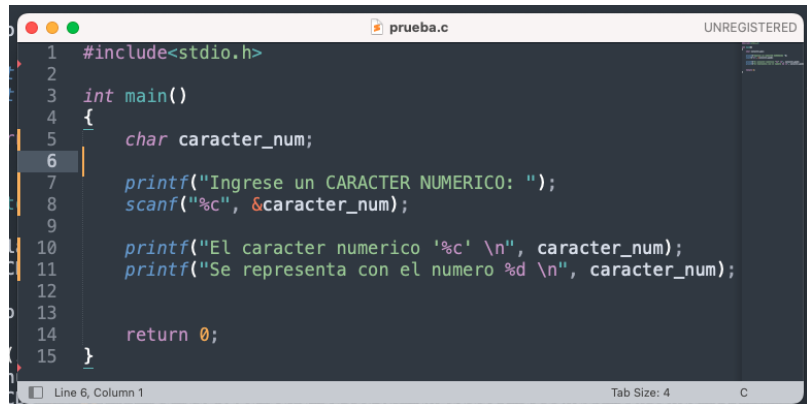

Ejercicio 2: Debe realizar una función que reciba un carácter numérico y retorne un número entero.

Las computadoras solo entienden números, por lo que cuando se escriben caracteres (char) estos tienen una representación numérica basada en ASCII.

Esto quiere decir que por ejemplo al ingresar el carácter 'a' la computadora lo que recibe es 97 y es en un segundo momento, cuando se realiza la salida del flujo de datos por pantalla, en que se le asigna la letra.

De igual manera sucede con los números. Una cosa es el **CARACTER** '1' y otra el **NUMERO** 1. El carácter '3' se almacena en una variable de tipo char, que al imprimirla en pantalla se ve como 3. Sin embargo, esta variable char no es más que un número, específicamente el carácter '3' tiene un valor de 51 en la tabla ASCII.

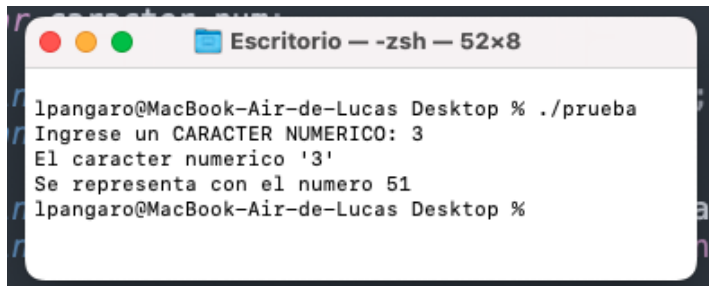
En este programa de prueba podemos ver como una misma variable de tipo char puede imprimir tanto su "valor interno" como la representación del mismo.



```
1 #include<stdio.h>
2
3 int main()
4 {
5     char caracter_num;
6
7     printf("Ingrese un CARACTER NUMERICO: ");
8     scanf("%c", &caracter_num);
9
10    printf("El caracter numerico '%c' \n", caracter_num);
11    printf("Se representa con el numero %d \n", caracter_num);
12
13
14    return 0;
15 }
```

Utilizando el mismo ejemplo, se le asigna a `caracter_num` el carácter 3. Al hacer un `printf` con `"%c"` la terminal muestra el carácter '3' mientras que al imprimirlo con

`"%d"` podemos ver el valor que tiene dicho carácter internamente (51).



```
lpangaro@MacBook-Air-de-Lucas Desktop % ./prueba
Ingrese un CARACTER NUMERICO: 3
El caracter numerico '3'
Se representa con el numero 51
lpangaro@MacBook-Air-de-Lucas Desktop %
```

Guiándonos con la siguiente tabla sabemos que si el carácter '0' tiene un valor de 48 para transformarlo en un entero simplemente debemos restar 48 al valor del char.

Además de tener en cuenta esta propiedad de los char y su representación, para la resolución de este ejercicio tenemos que saber algo elemental de las matemáticas, esto es el **valor posicional**.

En un número, por ejemplo, el 234 sabemos que el 4 representa las unidades, el 3 las decenas y el 2 las centenas.

Si descomponemos este número obtendremos $2 \times 100 + 3 \times 10 + 4 \times 1$.

**TABLA ASCII DE
NUMEROS**

'0'	= 48
'1'	= 49
'2'	= 50
'3'	= 51
'4'	= 52
'5'	= 53
'6'	= 54
'7'	= 55
'8'	= 56
'9'	= 57

elcodigoascii.com.ar

Ahora sí, algunos detalles del código:

```

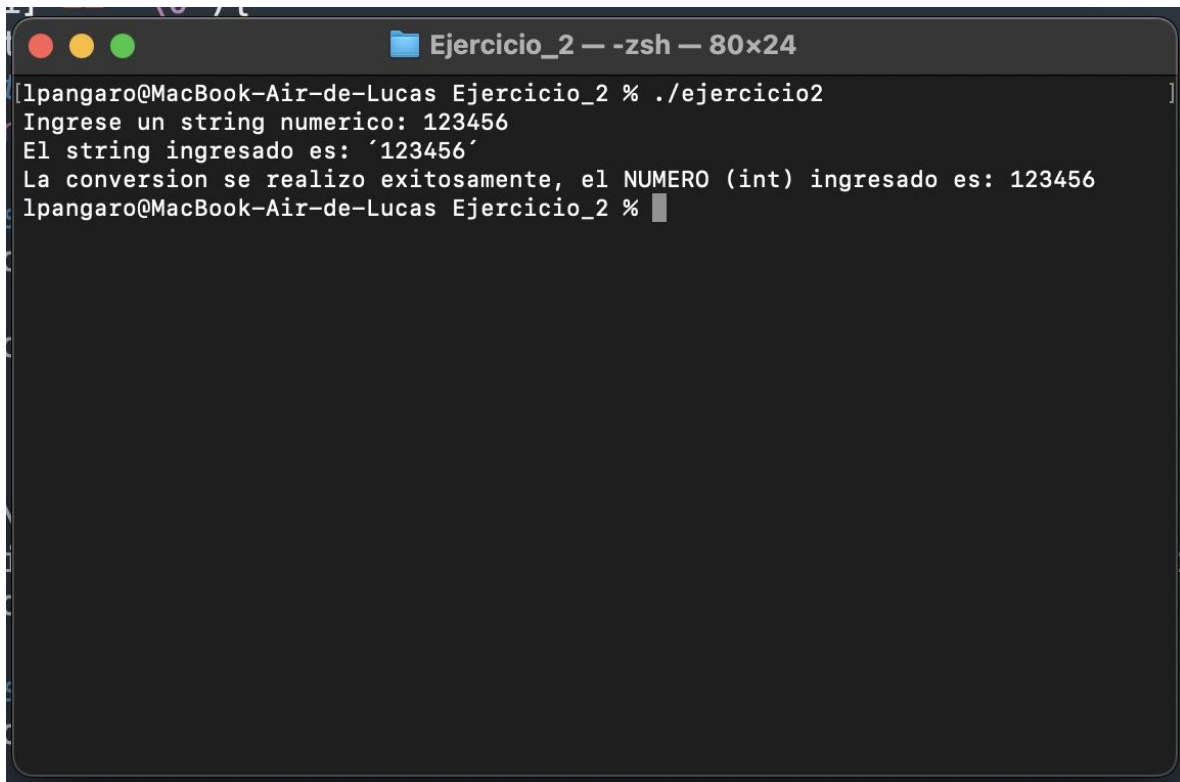
26 #define diferenciaASCII 48
48
49 int my_atoi(char* string){
50
51     int largo, i, numero = 0;
52     int VP = 1;    // Valos posicional 1, 10, 100, 1000 ...
53
54     largo = strlen(string);
55
56     for (i = (largo-1) ; i >= 0 ; i--) { // uso largo-1 porque el indice del vector comienza en 0
57         numero += (string[i] - diferenciaASCII) * VP;
58         VP *= 10; // Se utiliza para calcular unidades, decenas, centenas ...
59     }
60
61     return numero;
62 }

```

Podemos destacar dos partes de este algoritmo, una es la resta al carácter numérico char para obtener el valor int. Se utilizo un #define para dotar de mayor expresividad al código.

Y la otra mención importante es sobre el valor posicional. Al crear el ciclo se comenzó de atrás hacia delante, de esta forma el primer número se multiplica por 1 (unidades), el segundo por 10 (decenas), el tercero por 100 (centenas).

En cada vuelta la variable VP se multiplica por 10 y el numero calculado finalmente se retorna al main.



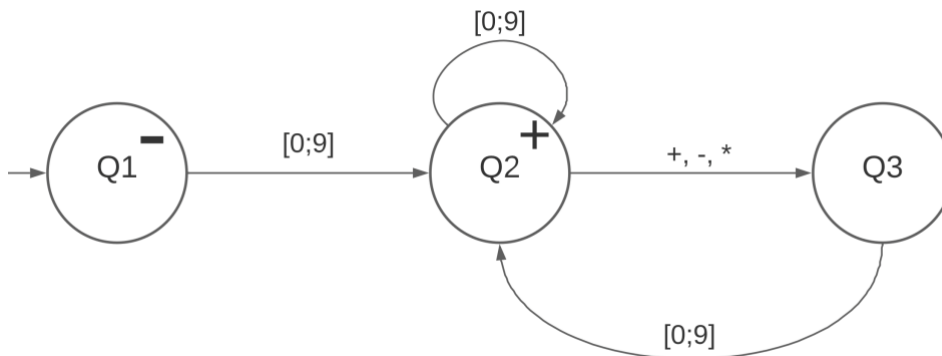
```
Ejercicio_2 — -zsh — 80x24
lpangaro@MacBook-Air-de-Lucas Ejercicio_2 % ./ejercicio2
Ingrese un string numerico: 123456
El string ingresado es: '123456'
La conversion se realizo exitosamente, el NUMERO (int) ingresado es: 123456
lpangaro@MacBook-Air-de-Lucas Ejercicio_2 %
```

Ejercicio 3: Ingresar una cadena que represente una operación simple con enteros decimales y obtener su resultado, se debe operar con +, _ y *. Ejemplo $3+4*7+3-5 = 29$. Debe poder operar con cualquier número de operandos y operadores respetando la precedencia de los operadores aritméticos.

La cadena ingresada debe ser validada previamente preferentemente reutilizando las funciones del ejercicio 1.

Para poder realizar la operación los caracteres deben convertirse a números utilizando la función 2.

La cadena debe ingresar por línea de comando o por archivo.



	0	1	2	3	4	5	6	7	8	9	+	-	*
Q1 -	Q2	Q2	Q2	Q2	Q2	Q2	Q2	Q2	Q2	Q2	Q4	Q4	Q4
Q2 +	Q2	Q2	Q2	Q2	Q2	Q2	Q2	Q2	Q2	Q2	Q3	Q3	Q3
Q3	Q2	Q2	Q2	Q2	Q2	Q2	Q2	Q2	Q2	Q2	Q4	Q4	Q4
Q4	Q4	Q4	Q4	Q4	Q4	Q4	Q4	Q4	Q4	Q4	Q4	Q4	Q4

Antes de comenzar con la resolución del ejercicio se explicará el razonamiento utilizado.

Se parte de un vector de caracteres que denominaremos cadena. La misma tiene el siguiente formato:

```
char cadena [1000];
```

cadena →

'1'	'2'	'+'	'3'	'+'	'4'	'-'	'5'	'+'	'6'	'*'	'7'	'\0'				
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	------	--	--	--	--

Una función que se desarrolló se encarga de diferenciar los números de los operadores y separarlos en distintos vectores. Para ello utilizamos `v_num` y `v_ope`.

`int v_num [1000]` `v_num` →

12	3	4	5	6	7				
----	---	---	---	---	---	--	--	--	--

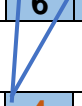
`char v_ope[1000]` `v_ope` →

'+'	'+'	'-'	'+'	'*'	'\0'				
-----	-----	-----	-----	-----	------	--	--	--	--

Ahora bien, si vemos en detalle podemos apreciar cierta relación entre los índices de ambos vectores. Por ejemplo, el operador `'*'` se encuentra en la posición (4) de `v_ope`. Lo que debería hacer es operar el 6 con el 7; que tienen el índice (4) y (5) respectivamente en `v_num`.

<code>v_num</code> →	Índice	0	1	2	3	4	5	6				999
		12	3	4	5	6	7					

<code>v_ope</code> →	Índice	0	1	2	3	4	5	6				999
		'+'	'+'	'-'	'+'	'*'	'\0'					



Ahora el signo menos `'-'` opera con los números 4 y 5. El índice del `'-'` en `v_ope` es (2) y del 4 y 5 en `v_num` el (2) y (3). **Es decir, cada operador se relaciona con dos números, y los índices de esos números son: el mismo que el del operador y el siguiente.**

Más ejemplos:

<code>v_num</code> →	Índice	0	1	2	3	4	5	6				999
		12	3	4	5	6	7					

<code>v_ope</code> →	Índice	0	1	2	3	4	5	6				999
		'+'	'+'	'-'	'+'	'*'	'\0'					

Para sumar el 12 y el 3 los índices serían:

En el vector de operadores: 0 del `'+'`
 En el vector de números: 0 del 12
 1 del 3

v_num →

Índice	0	1	2	3	4	5	6				999
	12	3	4	5	6	7					

v_ope →

Índice	0	1	2	3	4	5	6				999
	'+'	'+'	'-'	'+'	'*'	'\0'					

Para sumar el 3 y el 4 los índices serían:

En el vector de operadores: 1 del '+'

En el vector de números: 1 del 3
2 del 4

```
if(v_ope[i] == '+'){
    v_num[i] = v_num[i] + v_num[i+1];
}
```

En todos los casos coincide el índice del operador con los índices de los números que debe operar (el mismo y el siguiente).

Ahora bien, cuando se realiza una operación podemos decir que los números se consumen. Dentro de nuestra función vamos realizando semi-resultados de la cadena ingresada. Cada uno de estos semi-resultados, o resultados parciales, se guarda dentro del vector de números. ¿En que posición? En la del índice de la operación, es decir, en el índice del primer dígito que estoy operando; y para no repetir números ponemos un 0 en la posición del otro de los números. Por ejemplo

v_num →

Índice	0	1	2	3	4	5	6				999
	12	3	4	5	6	7					

v_ope →

Índice	0	1	2	3	4	5	6				999
	'+'	'+'	'-'	'+'	'*'	'\0'					

$$3+4=7$$

v_num →

Índice	0	1	2	3	4	5	6				999
	12	7	0	5	6	7					

Una vez que se realizan todas las operaciones del vector v_ope nos queda v_num con todos los resultados parciales; Lo único que queda por realizar es la suma de todos ellos.

Pero antes de ver en detalle el código vamos a mostrar un poco más de este funcionamiento.

“12+3+4-5+6*7”

v_ope →

Índice	0	1	2	3	4	5	6				999
	'+'	'+'	'-'	'+'	'*'	'\0'					

v_num →

Índice	0	1	2	3	4	5	6				999
	12	3	4	5	6	7					

Vamos a realizar el producto. Para ello vemos el índice del carácter '*' (4) y multiplicamos el número que se encuentre en la posición (4) con el de la siguiente (5). En este caso 6*7

v_num →

Índice	0	1	2	3	4	5	6				999
	12	3	4	5	6*7 = 42	0					

Se resuelve la operación (6*7). Su resultado se guarda en donde antes estaba el 6 (primer termino) y en el lugar del 7 (segundo término) se coloca un 0 para no volver a operar con dicho valor

Ahora continuamos realizando el mismo proceso con las sumas...

v_num →

Índice	0	1	2	3	4	5	6				999
	12+3 = 15	0	4	5	42	0					

i	0	1	2	3	4	5	
	'+'	'+'	'-'	'+'	'*'	'\0'	

v_num →

Índice	0	1	2	3	4	5	6				999
	15	0+4 = 4	0	5	42	0					

i	0	1	2	3	4	5	
	'+'	'+'	'-'	'+'	'*'	'\0'	

v_num →

Índice	0	1	2	3	4	5	6				999
	15	4	0-5 = -5	0	42	0					

i	0	1	2	3	4	5	
	'+'	'+'	'-'	'+'	'*'	'\0'	

v_num →

Índice	0	1	2	3	4	5	6				999
	15	4	-5	0+42 = 42	0	0					

i	0	1	2	3	4	5	
	'+'	'+'	'-'	'+'	'*'	'\0'	

v_num →

Índice	0	1	2	3	4	5	6				999
	15	4	-5	42	0	0					

Ahora recorriendo este vector y sumando cada uno de sus elementos obtenemos el resultado de la operación escrita en la cadena!

Luego de esa explicación comencemos con algunas menciones especiales sobre el código.

Al igual que en el punto 1 se tiene un main bastante breve donde se da a elegir si se quiere ingresar la cadena por línea de comandos o por un archivo.

```

37
38     switch (menu){
39         case 1 :
40             printf("Ingrese la cadena de caracteres numericos (Hexadecimales, Octales o Decimales): ");
41             scanf ("%s", cadena);
42             break;
43
44         case 2 :
45             printf("Ingrese el nombre del archivo (ej: entrada.txt): ");
46             scanf ("%s", name_file);
47             ingreso_por_archivo(name_file, cadena);
48             break;
49
50         default:
51             printf ("No ingreso una opcion valida \n");
52             return 1;
53     }
54

```

Posteriormente se valida la cadena ingresada, para ello se utilizó el autómata mostrado al inicio. Este acepta las cadenas que comienzan y terminan con un carácter numérico y en el medio puede contener números y operadores*, sin embargo, si es un operador debe estar ante- y pre-cedido por un número. Esto hace que la cadena “*1+2”, “1+2*” o “1++2” no sean reconocidas.

(*operadores: '+', '*', '-').

La función `isoperator` también fue desarrollada y únicamente recibe un char, pregunta si es '+', '-', o '*' y devuelve 1 para casos afirmativos y 0 en caso de no ser ninguno de los operadores.

```

136 //AUTOMATA
137 int validar_vector(char* cadena) {
138     int estado = INICIO, i=0;
139     while(1){
140         switch(estado){
141             case INICIO: //Q1
142                 if(isdigit(cadena[i]))
143                     estado = NUMERO_O_OPERADOR;
144                 else
145                     estado = RECHAZO;
146                 i++;
147                 break;
148
149             case NUMERO_O_OPERADOR: //Q2
150                 if (cadena[i] == '\0')
151                     return True; //valido
152                 else if(isdigit(cadena[i]))
153                     estado = NUMERO_O_OPERADOR;
154                 else if(isoperator(cadena[i]))
155                     estado = NUMERO; //Despues de un
156                 else
157                     estado = RECHAZO;
158                 i++;
159                 break;
160
161             case NUMERO: //Q3
162                 if(isdigit(cadena[i]))
163                     estado = NUMERO_O_OPERADOR;
164                 else
165                     estado = RECHAZO;
166                 i++;
167                 break;
168
169             case RECHAZO:
170                 return False; //invalido
171         }
172     }
173 }

```

Una vez que la cadena está validada se separan los números de los operadores en `v_num` y `v_ope`. El prototipo de la función que realiza esto es

```
void separar_numeros_de_operadores(char* cadena, char* v_ope,
                                   int* v_num, int* largo)
```


No hay mucho que explicar sobre esta función más que mencionar que se le pasa por referencia la variable `largo` para saber la cantidad de elementos que tiene el vector de números.

```

55     if (validar_vector(cadena)){
56
57         separar_numeros_de_operadores(cadena, v_ope, v_num, &largo);
58         resultado = calcular(v_ope, v_num, largo);
59         printf("El Resultado es: %d \n", resultado);
60     }

```

Una vez separados los números de los operadores se pasan ambos vectores a la función `calcular`. Esta devuelve el resultado de la cadena ingresada anteriormente.

```

97 int calcular (char* v_ope, int* v_num, int largo) {
98     int i, resultado = 0;
99
100    for(i=(strlen(v_ope) - 1); i >= 0; i--){ // recorro de atras para adelante por si tengo multiples productos no multiplicar por 0
101        if(v_ope[i] == '*'){ //Primero resuelvo el * para respetar precedencia
102            v_num[i] = v_num[i] * v_num[i+1];
103            v_num[i+1] = 0;
104        }
105    }
106
107    for(i=0; i < strlen(v_ope); i++){
108        if(v_ope[i] == '+'){
109            v_num[i] = v_num[i] + v_num[i+1];
110            v_num[i+1] = 0;
111        }
112        if(v_ope[i] == '-'){
113            v_num[i] = v_num[i] - v_num[i+1];
114            v_num[i+1] = 0;
115        }
116    }
117
118    for(i=0; i < largo; i++){
119        resultado += v_num[i];
120    }
121
122    return resultado;
123 }

```

Si vemos más de cerca las líneas 100 y 101 son muy importantes.

```

98     int i, resultado = 0;
99
100     for(i=(strlen(v_ope) - 1); i >= 0; i--){
101         if(v_ope[i] == '*'){
102             v_num[i] = v_num[i] * v_num[i+1];
103             v_num[i+1] = 0;
104         }
105     }

```

Se debe comenzar preguntando por el operador “*” para respetar la precedencia de operadores.

Y el otro detalle importante es que se recorre de atrás hacia adelante. ¿por qué?, Porque como vimos anteriormente cuando se resuelve una operación guardamos el valor del resultado en el índice del primer número y colocamos un cero en el índice

siguiente. Si tenemos una multiplicación sucesiva y se recorre de izquierda a derecha el siguiente producto se realizaría con valor 0.

Por ejemplo: $2*3*4 = 24$

Si recorremos de izquierda a derecha

```
if(v_ope[i] == '*'){
    v_num[i] = v_num[i] * v_num[i+1];
    v_num[i+1] = 0;
}
```

v_ope →

Índice	0	1			
	*	*			

v_num →

Índice	0	1	2		
	2	3	4		

v_num →

Índice	0	1	2		
	2*3=6	0	4		

Al poner 0 en la posición siguiente hacemos que el producto que sigue realice $0*4=0$

v_num →

Índice	0	1	2		
	6	$0*4=0$	0		

Mientras que si lo recorremos de derecha a izquierda:

v_ope →

Índice	0	1			
	*	*			

v_num →

Índice	0	1	2		
	2	3	4		

v_num →

Índice	0	1	2		
	2	$3*4=12$	0		

v_num →

Índice	0	1	2		
	$2*12=24$	0	0		

Por esta razón el bucle del "*" se realiza de atrás para delante.

```

107     for(i=0; i < strlen(v_ope); i++){
108         if(v_ope[i] == '+'){
109             v_num[i] = v_num[i] + v_num[i+1];
110             v_num[i+1] = 0;
111         }
112         if(v_ope[i] == '-'){
113             v_num[i] = v_num[i] - v_num[i+1];
114             v_num[i+1] = 0;
115         }
116     }
117
118     for(i=0; i < largo; i++){
119         resultado += v_num[i];
120     }
121
122     return resultado;

```

A continuación, se analizan los siguientes operadores ('+', '-'). Si es un más se hace la suma del número y el siguiente; se “limpia” esta última posición. Si es un menos se realiza la resta y también se “limpia” el valor siguiente.

Finalmente, un `for` recorre el vector de enteros sumando los resultados y se retorna dicho valor.

```

Ingrese la cadena de caracteres numericos (Hexadecimales, Octales o Decimales): 3+4*7+3-5
El Resultado es: 29
lpangaro@MacBook-Air-de-Lucas Ejercicio_3 %

```

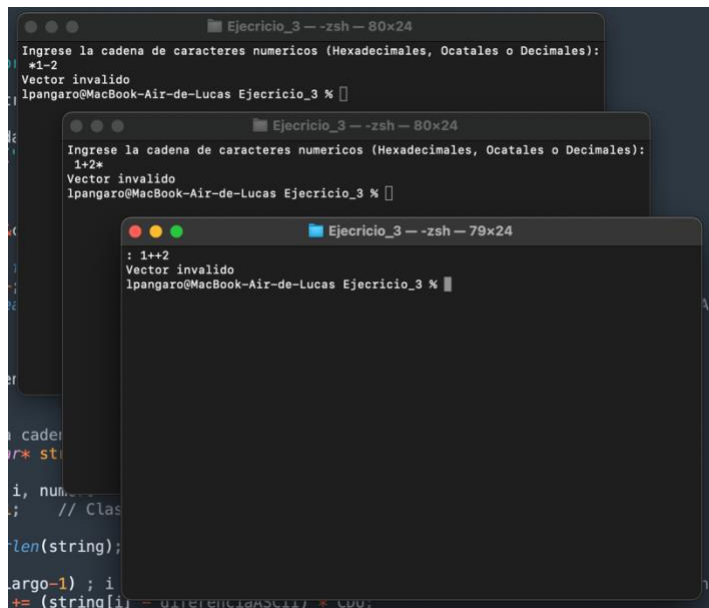
Programa corriendo con ingreso por línea de comando.

```

Ingrese el nombre del archivo (ej: entrada.txt): entrada.txt
El Resultado es: 29
lpangaro@MacBook-Air-de-Lucas Ejercicio_3 %

```

Mismo programa pero corrido con un archivo .txt como entrada



The image shows three overlapping terminal windows titled 'Ejercicio_3 - zsh - 80x24'. Each window displays the same sequence of commands and output:

```
Ingrese la cadena de caracteres numericos (Hexadecimales, Octales o Decimales):  
*1-2  
Vector invalido  
lpangaro@MacBook-Air-de-Lucas Ejercicio_3 %
```

The third window also shows the beginning of a C++ code snippet at the bottom:

```
...  
; i+=2  
// Clas  
len(string);  
largo-1) ; i  
+= (string[i] - 0110100011) * 100;
```

Prueba de validación del vector:

1er caso: operador al comienzo de la cadena.

2do caso: Operador al final de la cadena.

3er caso: Operador repetido.