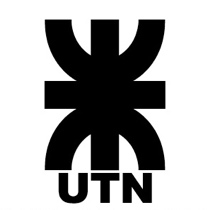
******

***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 Gianlucca | 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 numero (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.

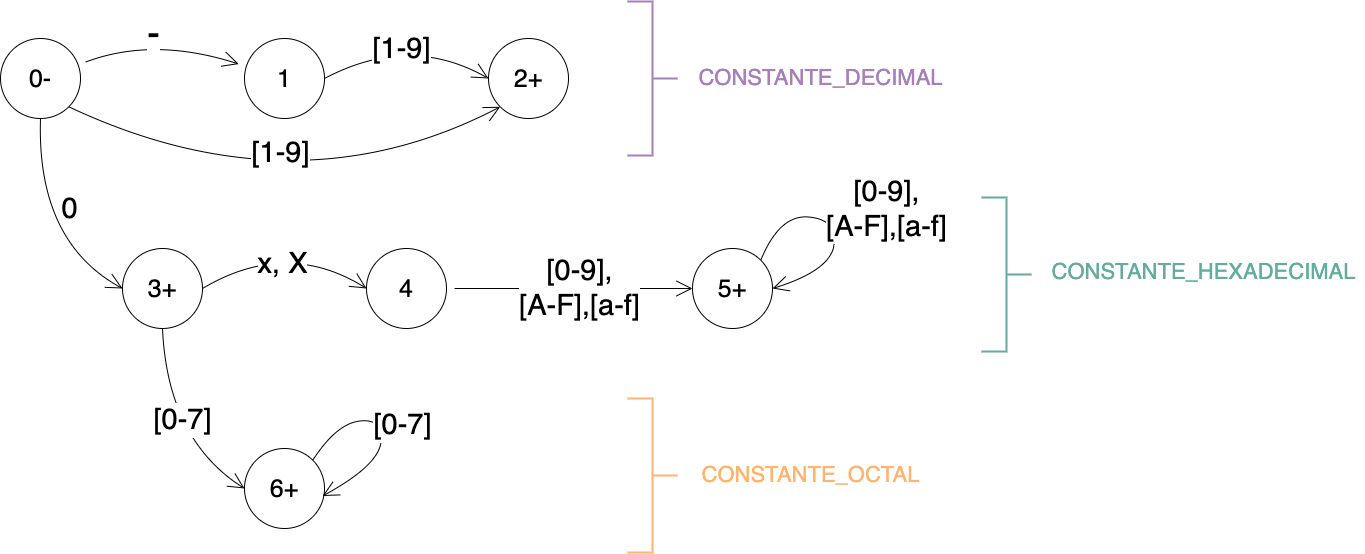
Algunos ejemplos son: 012, 0034, 076.

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 numero será un octal y quedaron en ese estado preguntando por los siguientes, salvo que continue 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 numero 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 validos son: 123, -45, 7, -9, 10.

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

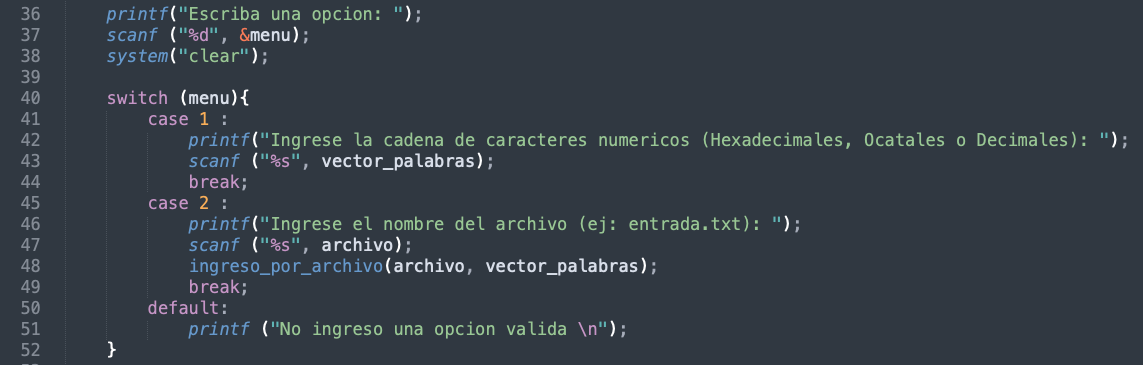


|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | - | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a, A | b, B | c, C | d, D | e, E | f, F | x, X |
| 0- | 1 | 3 | 2 | 2 | 2 | 2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

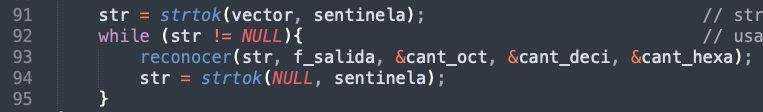
****

Cualquiera sea la opción elegida se guarda la cadena en el vector\_palabras que luego es enviado a la función separar\_y\_reconocer.

****

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.

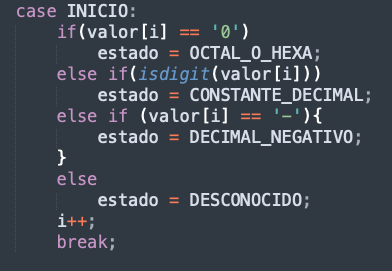


La función reconocer es la más compleja de todas, es la encargada de recorrer cada sub-cadena y decir a que tipo de constante pertenece.

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.

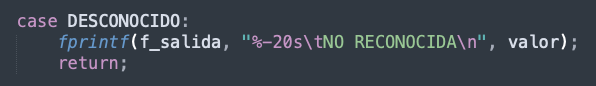
Si este carácter es un 0 es posible que el número leído sea un octal o un hexadecimal. Es por esto que el estado cambia a OCTAL\_O\_HEXA.

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.

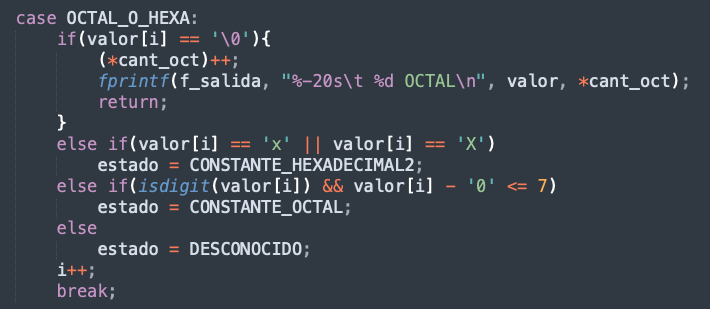
Si el estado llega a desconocido se imprime que la cadena no fue reconocida y termina el programa



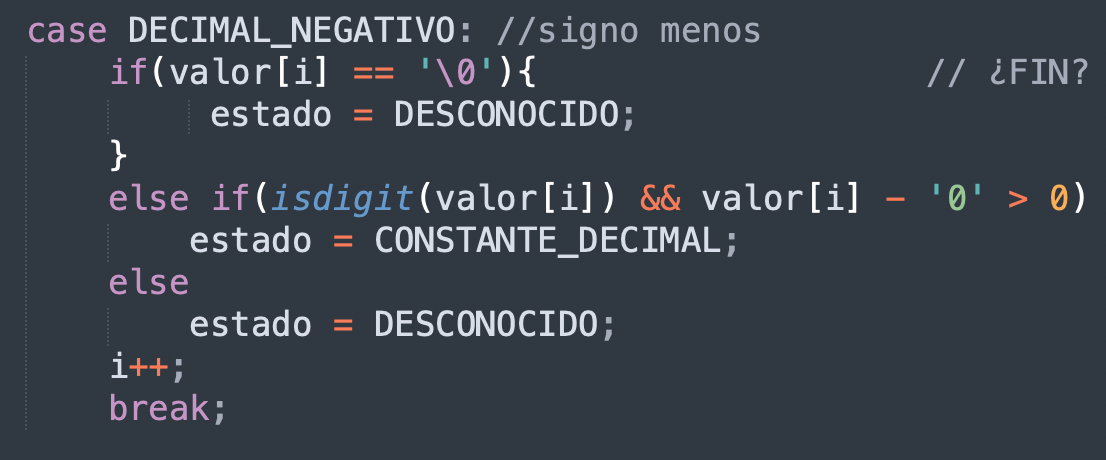
Continuando en orden y recordando si el carácter era un ‘0’ se enviaba al estado OCTAL\_O\_HEXA.

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).



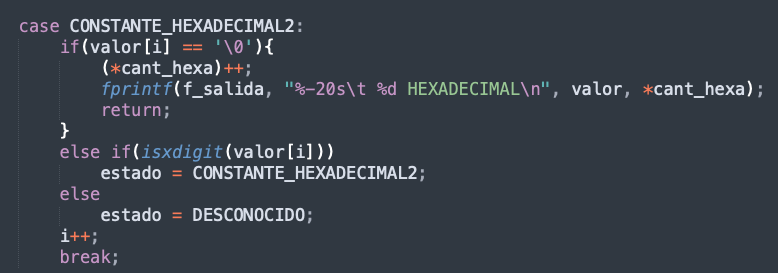
Otro estado especial es el DECIMAL\_NEGATIVO a este entramos cunado el primer carácter de la sub-cadena era un ‘-‘, ahora preguntamos si el carácter que le sigue es un digito salvo el 0 (porque no tendría sentido tener -0) si es un digito, el estado cambia a CONSTANTE\_DECIMAL

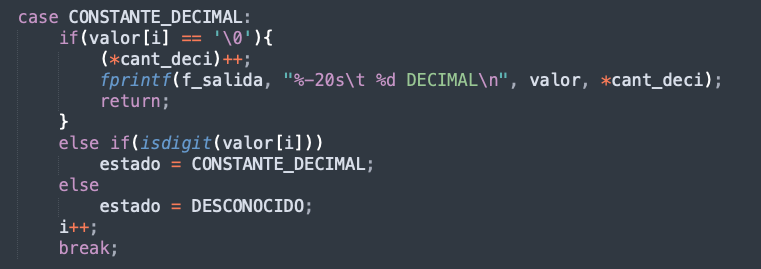


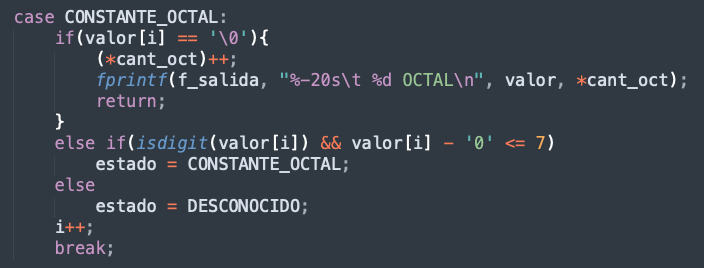
Para los siguientes estados es siempre lo mismo, CONSTANTE\_OCTAL, CONSTANTE\_DECIMAL y CONSTANTE\_HEXADECIMAL, realizan tres preguntas. Si estoy en ese estado y **llega el carácter ‘\0**’ quiere decir que la sub-cadena termino, por lo que se guarda en un archivo de salida el valor de la constante, el 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.

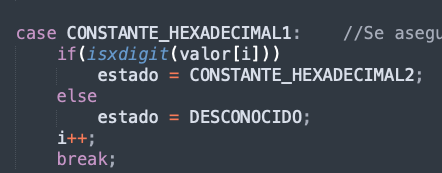
Tercero y por último si no cumple con ninguna de las anteriores quiere decir que la cadena NO ES RECONCOIDA, el estado cambia a DESCONOCIDO.

****

****

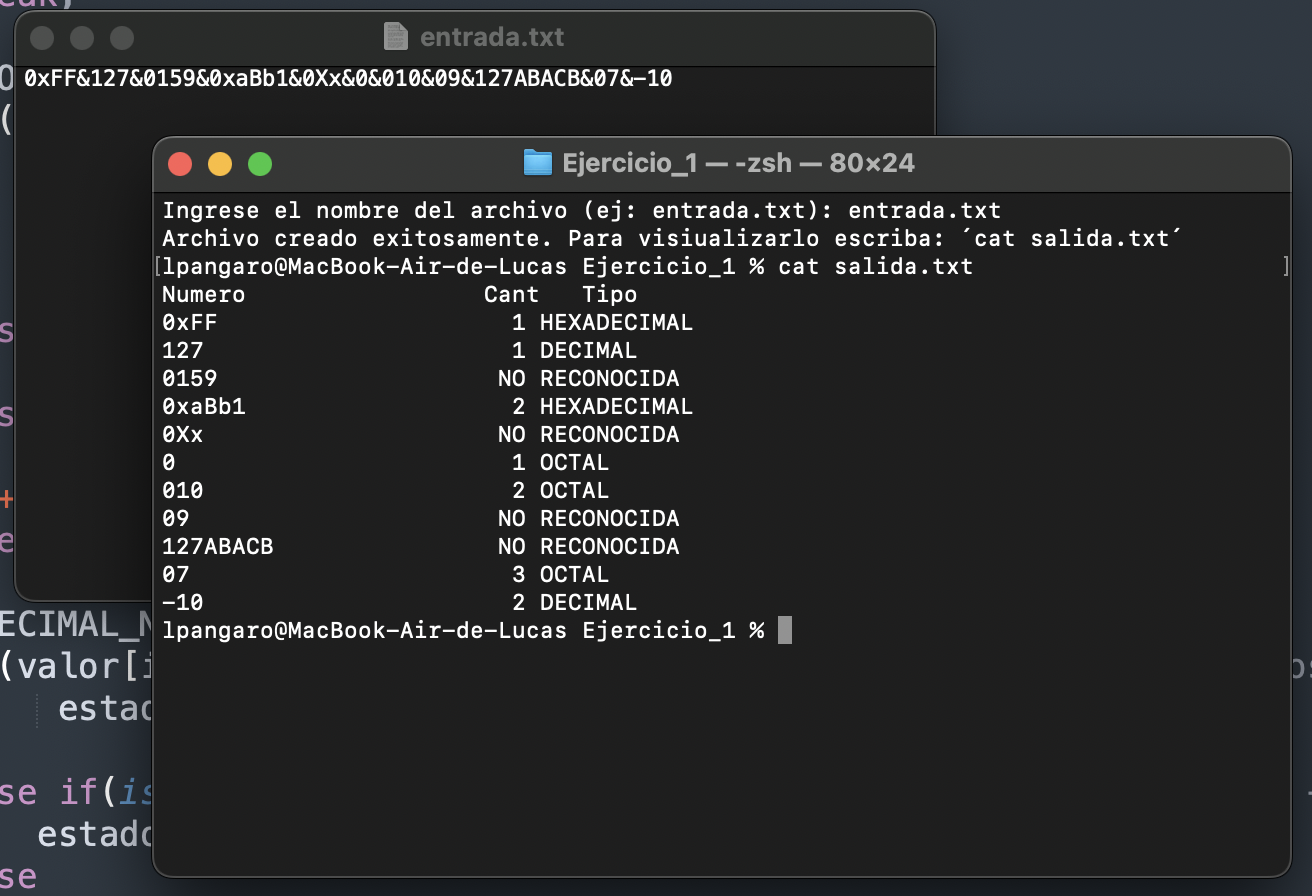
****

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 continue con por lo menos otro valor. De esta manera no permite las cadenas “0x” o “0X” como hexadecimal.



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 continue 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



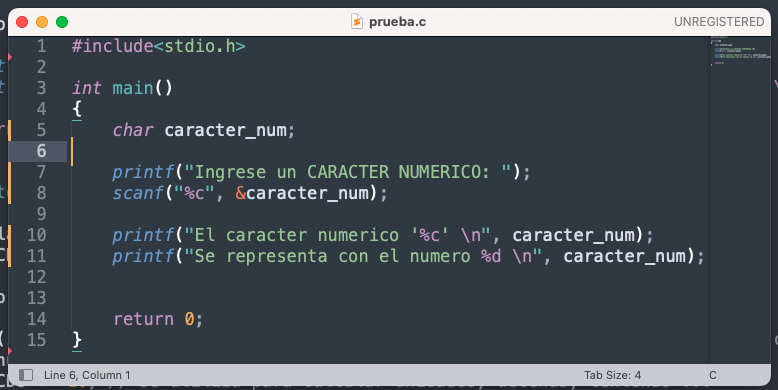
**Ejercicio 2:** Debe realizar una función que reciba un carácter numérico y retorne un número entero.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

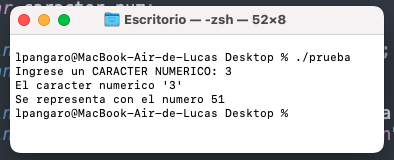
Las computadoras solo entienden números, por lo que cuando se escriben caracteres (char) estos tienen una representación numérica basada en ASCII.

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.

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).

Guiándonos con la siguiente tabla sabemos que si el carácter

**TABLA ASCII DE NUMEROS**

´0´ = 48

´1´ = 49

´2´ = 50

´3´ = 51

´4´ = 52

´5´ = 53

´6´ = 54

´7´ = 55

´8´ = 56

´9´ = 57

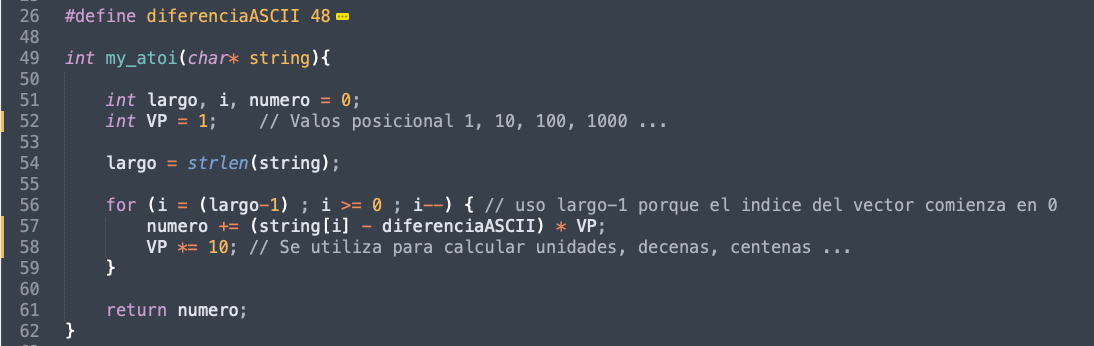
elcodigoascii.com.ar

‘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.

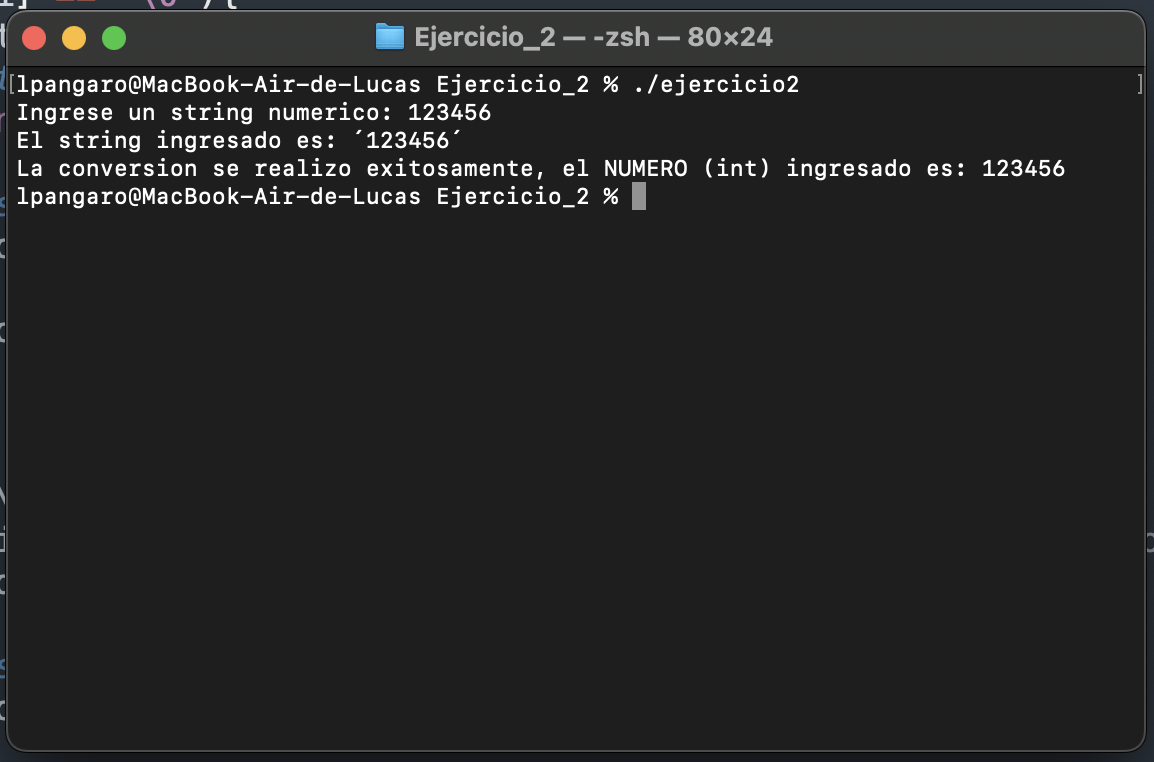
Si descomponemos este número obtendremos 2 x 100 + 3 x 10 + 4 x 1.

Ahora sí, algunos detalles del código: 

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.

****

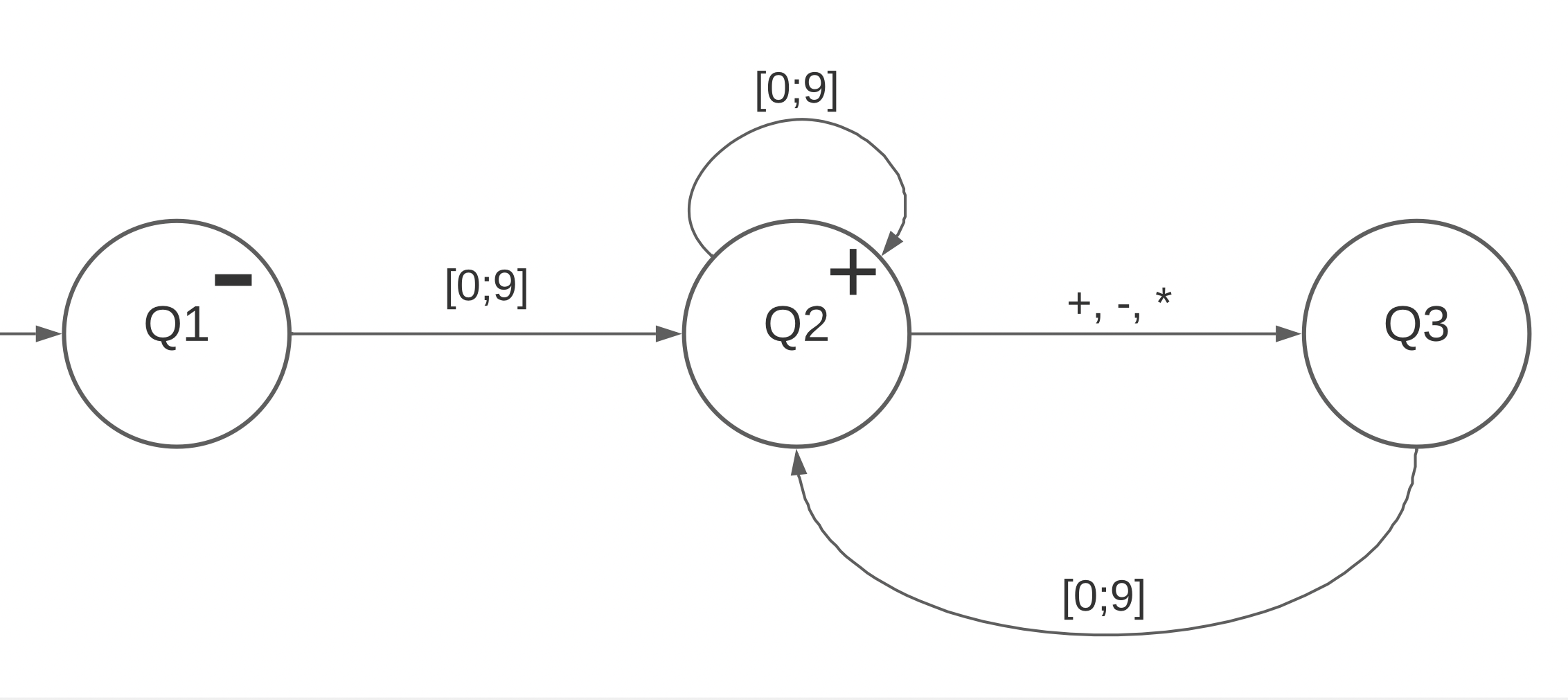
**Ejercicio 3:** *Ingresar una cadena que represente una operación simple con enteros decimales y obtener su resultado, se debe operar con +,\_ y \*. Ejemplo 3+4\*7+3-5 = 29. Debe poder operar con cualquier número de operandos y operadores respetando la precedencia de los operadores aritméticos.*

*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.

v\_num →

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

int v\_num [1000]

v\_ope →

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ‘+’ | ‘+’ | ‘-‘ | ‘+’ | ‘\*’ | ‘\0’ |  |  |  |  |  |

char v\_ope[1000]

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.

v\_num →

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Índice | 0 | 1 | 2 | 3 | **4** | **5** | 6 |  |  |  | 999 |
|  | 12 | 3 | 4 | 5 | **6** | **7** |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Índice | 0 | 1 | 2 | 3 | **4** | 5 | 6 |  |  |  | 999 |
|  | ‘+’ | ‘+’ | ‘-‘ | ‘+’ | **‘\*’** | ‘\0’ |  |  |  |  |  |

v\_ope →

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:

v\_num →

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Índice | **0** | **1** | 2 | 3 | 4 | 5 | 6 |  |  |  | 999 |
|  | **12** | **3** | 4 | 5 | 6 | 7 |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Índice | **0** | 1 | 2 | 3 | 4 | 5 | 6 |  |  |  | 999 |
|  | **‘+’** | ‘+’ | ‘-‘ | ‘+’ | ‘\*’ | ‘\0’ |  |  |  |  |  |

v\_ope →

Para sumar el 12 y el 3 los índices serian:

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 |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Índice | 0 | **1** | 2 | 3 | 4 | 5 | 6 |  |  |  | 999 |
|  | ‘+’ | **‘+’** | ‘-‘ | ‘+’ | ‘\*’ | ‘\0’ |  |  |  |  |  |

v\_ope →

Para sumar el 3 y el 4 los índices serian:

En el vector de operadores: 1 del ‘+’

En el vector de números: 1 del 3

2 del 4

En todos los casos coincide el índice del operador con los incides 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 digito 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 |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Índice | 0 | **1** | 2 | 3 | 4 | 5 | 6 |  |  |  | 999 |
|  | ‘+’ | **‘+’** | ‘-‘ | ‘+’ | ‘\*’ | ‘\0’ |  |  |  |  |  |

v\_ope →

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”

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Índice | 0 | 1 | 2 | 3 | **4** | 5 | 6 |  |  |  | 999 |
|  | ‘+’ | ‘+’ | ‘-‘ | ‘+’ | **‘\*’** | ‘\0’ |  |  |  |  |  |

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\_ope →

v\_num →

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

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

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

v\_num →

Ahora continuamos realizando el mismo proceso con las sumas…

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |  |  |  |  |  |

v\_num →

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | **4** | 5 |  |
|  | ‘+’ | ‘+’ | ‘-‘ | ‘+’ | **‘\*’** | ‘\0’ |  |

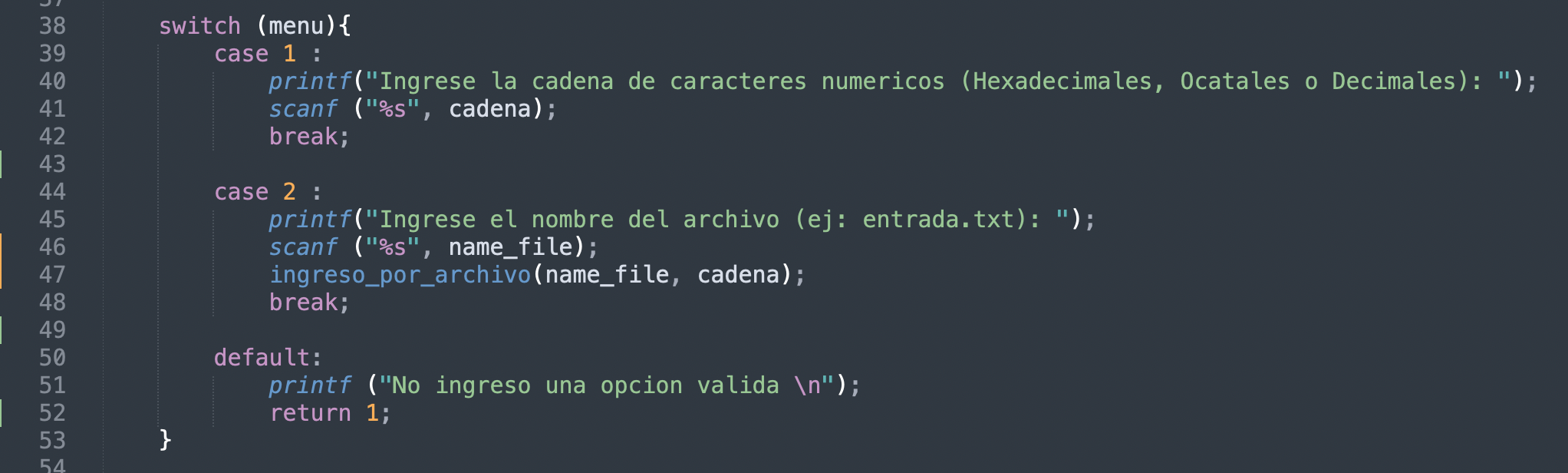
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Índice | 0 | 1 | 2 | 3 | 4 | 5 | 6 |  |  |  | 999 |
|  | 15 | 4 | -5 | 42 | 0 | 0 |  |  |  |  |  |

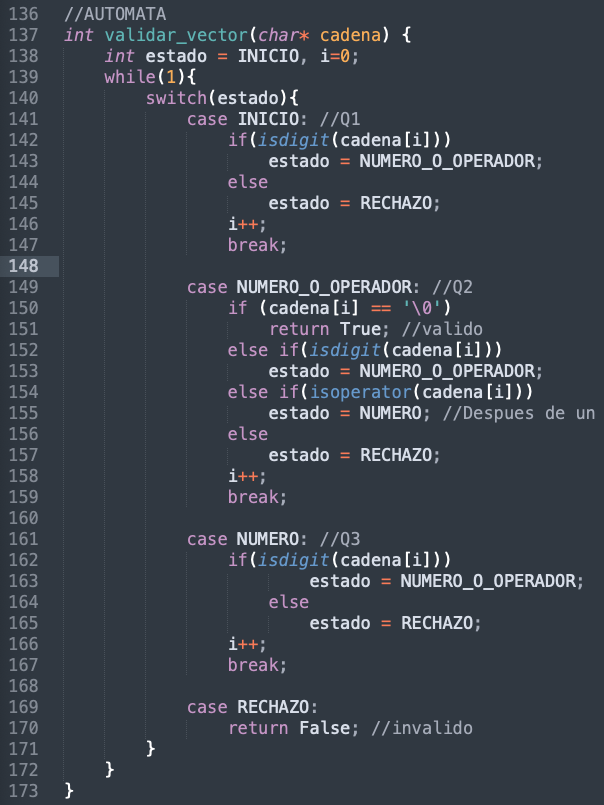
v\_num →

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.





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: ‘+’, ‘\*’, ‘-‘).

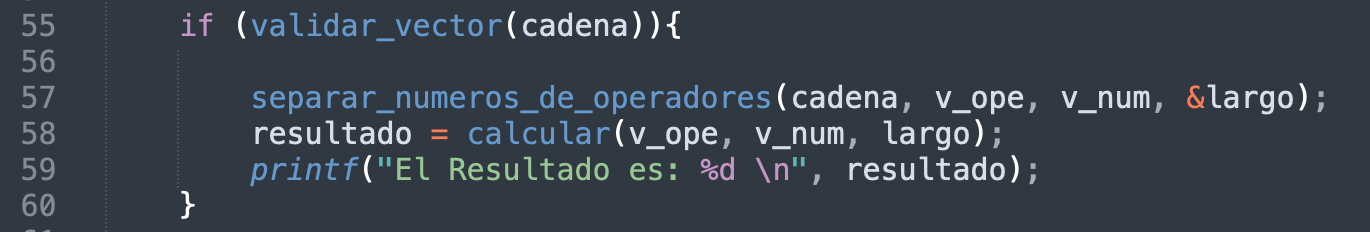
La función isoperator también fue desarrollada y únicamente recibe un char, pregunta si es ‘+’, ‘-‘, o ‘\*’ y devuelve 1 para casos afirmativos y 0 en caso de no ser ninguno de los operadores.

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

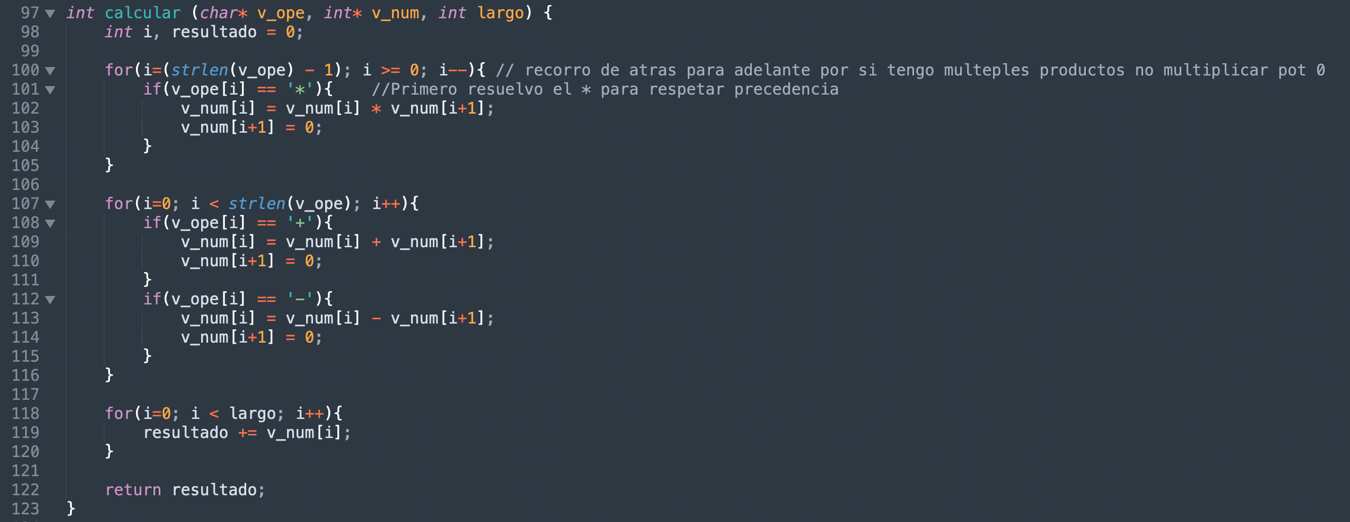
void separar\_numeros\_de\_operadores(char\* cadena, char\* v\_ope,

int\* v\_num, int\* largo)

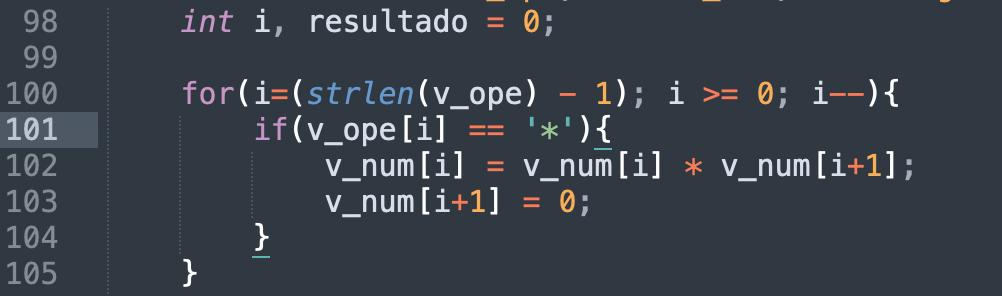
No hay mucho que explicar sobre esta función más que mencionar que se le pasa por referencia la variable largo para saber la cantidad de elementos que tiene el vector de números.



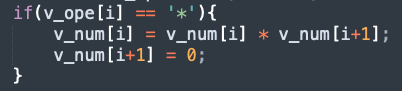
Una vez separados los números de los operadores se pasan ambos vectores a la función calcular. Esta devuelve el resultado de la cadena ingresada anteriormente.



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

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 delante. ¿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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Índice | 0 | 1 |  |  |  |
|  | ‘\*’ | ‘\*’ |  |  |  |

v\_ope →

v\_num →

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Índice | 0 | 1 | 2 |  |  |
|  | 2 | 3 | 4 |  |  |

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

v\_num →

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Índice | 0 | 1 | 2 |  |  |
|  | 2\*3=6 | **0** | 4 |  |  |

v\_num →

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Índice | 0 | 1 | 2 |  |  |
|  | 6 | 0\*4=0 | 0 |  |  |

Mientras que si lo recorremos de derecha a izquierda:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Índice | 0 | 1 |  |  |  |
|  | ‘\*’ | ‘\*’ |  |  |  |

v\_ope →

v\_num →

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Índice | 0 | 1 | 2 |  |  |
|  | 2 | 3 | 4 |  |  |

v\_num →

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Índice | 0 | 1 | 2 |  |  |
|  | 2 | 3\*4=12 | 0 |  |  |

v\_num →

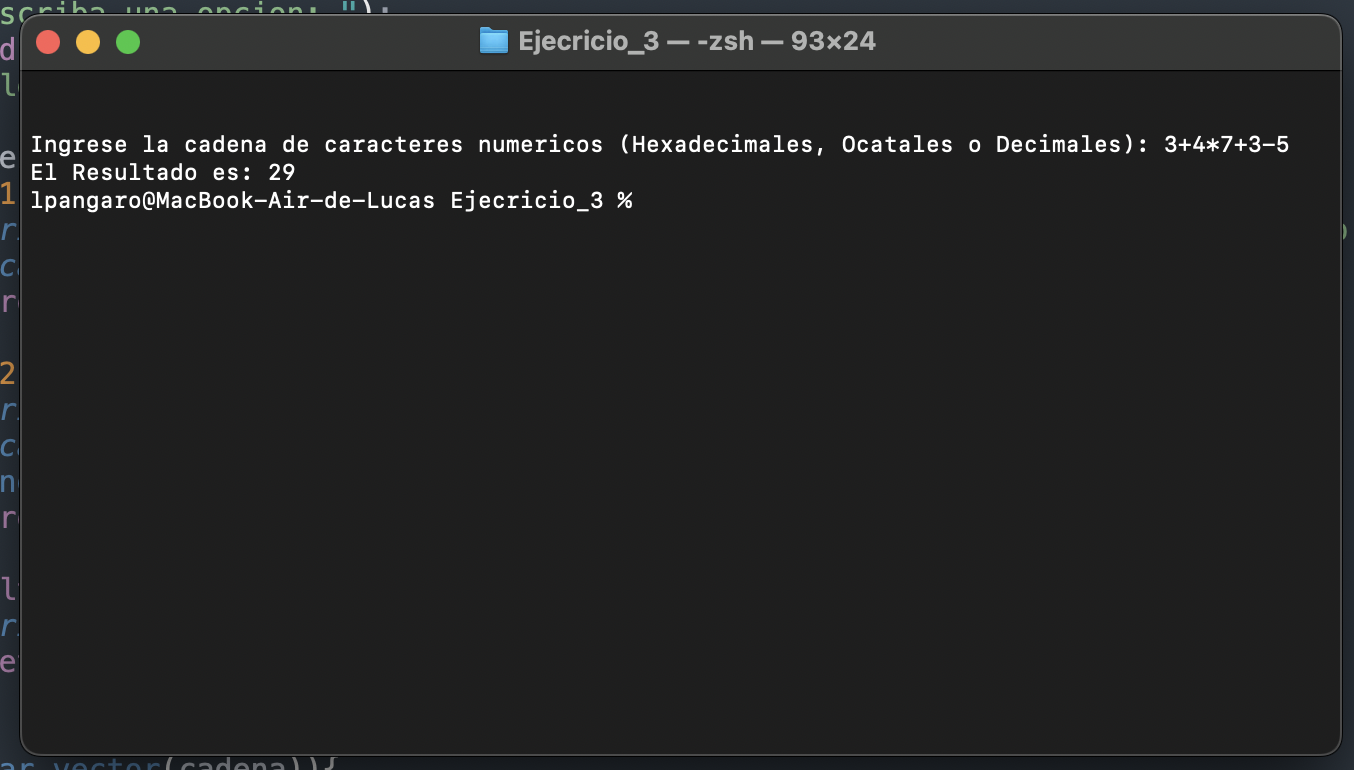
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Índice | 0 | 1 | 2 |  |  |
|  | 2\*12=24 | 0 | 0 |  |  |

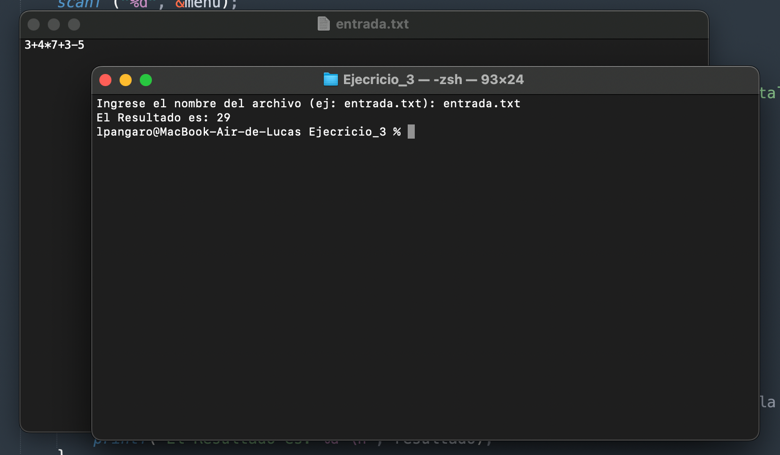
Por esta razón el bucle del “\*” se realiza de atrás para delante.



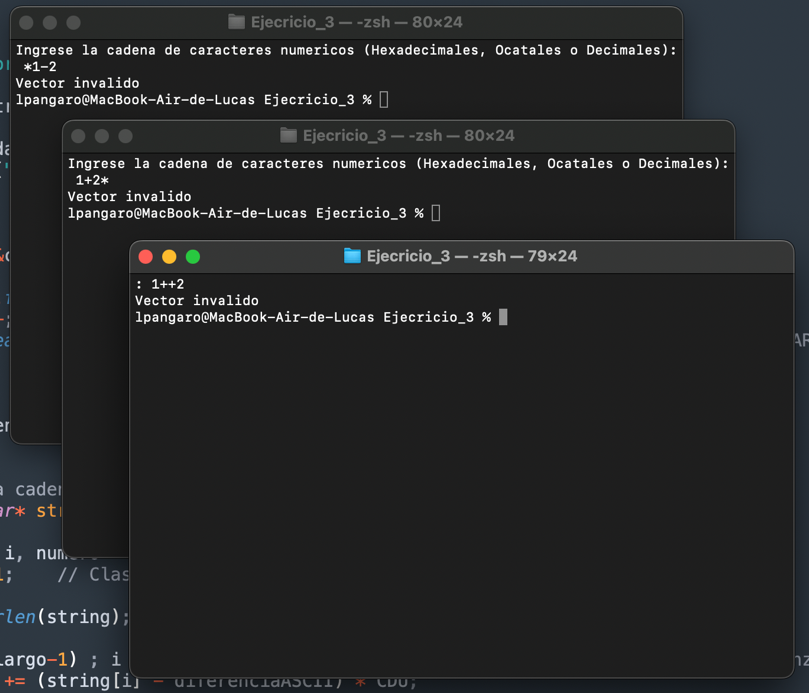
A continuación, se analizan los siguientes operadores (‘+’, ‘-’). Si es un más se hace la suma del número y el siguiente; se “limpia” esta última posición. Si es un menos se realiza la resta y también se “limpia” el valor siguiente.

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

Programa corriendo con ingreso por línea de comando.



Mismo programa pero corrido con un archivo .txt como entrada



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.