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.

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.

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'					

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					

v_ope ⟶

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

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					

v_ope ⟶

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

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					

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 numero 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 (primero termino) y en el lugar del 7 (segundo termino) 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					

v_num ⟶

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

v_num ⟶

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

v_num ⟶

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

```
38      switch (menu){
39          case 1 :
40              printf("Ingrese la cadena de caracteres numericos (Hexadecimales, Ocatales 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 realizó una función muy simple, pero con un detalle especial.

```c
int validar_vector(char* cadena) {

    int i = 0;

    while (cadena[i] != '\0') {
        if(!isdigit(cadena[i])) { // isdigit devuelve un valor no nulo si es un digito
            i++;
        }
        else if (isoperator(cadena[i]) && !isdigit(cadena[i - 1]) && !isdigit(cadena[i + 1])) {
            // SI el caracter leido es un operador tambien verifico que en la posicion siguiente y anterior haya un numero
            i++;
        }
        else
            return 1;
    }

    return 0;
}
```

Se recorre la cadena preguntando si el carácter leído es un digito o un operador. Pero 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.
La función `isoperator` también fue desarrollada y únicamente recibe un char, pregunta si es ‘+’, ‘-‘, o ‘*’ y devuelve 1 o 0.

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 se le pasa por referencia la variable `largo` para saber la cantidad de elementos que tiene el vector de números.

```c
    if (validar_vector(cadena)){

        separar_numeros_de_operadores(cadena, v_ope, v_num, &largo);
        resultado = calcular(v_ope, v_num, largo);
        printf("El Resultado es: %d \n", resultado);
    }
```

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 multeples productos no multiplicar pot 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      }
```

Sebe 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

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 ⟶

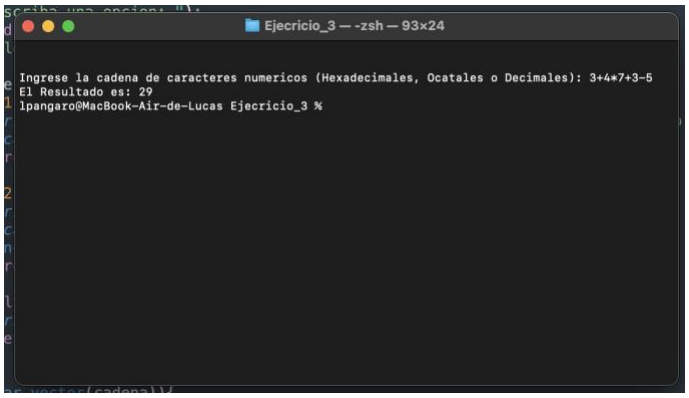
Índice	0	1	2		
	2*12=24	0	0		

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;
```

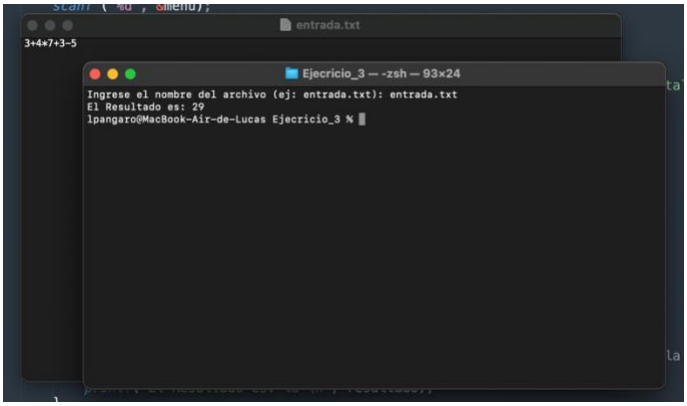
A continuación, se analizan los siguientes operadores ('+', '-'). Si es un más se hace la suma del número y el siguiente; y se "limpia" esta última posición. Si es un menos se realiza la resta ya también se "limpia" el valor siguiente.

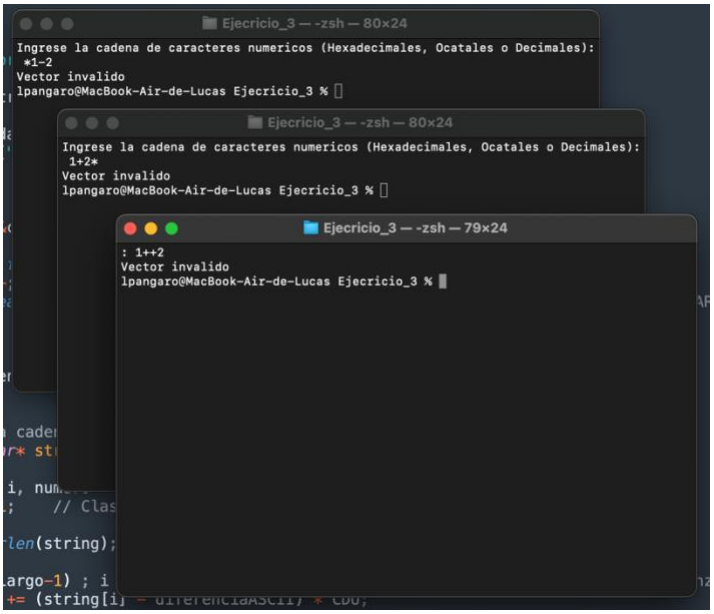
Finalmente, un for recorre el vector de enteros sumando los resultados y se retorna dicho valor.



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.