UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Sintaxis y Semántica de los Lenguajes
TPN2 - Autómatas

Integrantes - Grupo 7

ALUMNO	Legajo
Giovani Quispe	159978-1
Guido Biotti	171.434-0
Santiago Hamamura	203.717-8

Profesora Ing. Roxana Leituz

Curso K2006 - Sábados Turno mañana

2023

Enunciado

Todos los trabajos deben llevar carátula con todos los datos formales. Se deben incluir impresiones de las pantallas mostrando el funcionamiento del mismo.

La entrega debe contener los **archivos fuente** y un pdf con **las pantallas, instructivo y especificaciones** acerca de las decisiones tomadas para la resolución del mismo.

[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.

[2] Debe realizar una función que reciba un carácter numérico y retorne un número entero.

[3] Ingresar una cadena que represente una operación simple con enteros decimales y obtener su resultado, se debe operar con +,- ,/, *. Ejemplo = 3+4*8/2+3-5 = 17. Debe poder operar con cualquier número de operandos y operadores respetando la precedencia de los operadores aritméticos y sin paréntesis. 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.

El lenguaje de programación a utilizar debe ser C, no se aceptan trabajos en C++.

Para aprobar debe estar completo y funcionando correctamente, utilizar buenas prácticas de desarrollo e interfaces amigables.

Este trabajo puede requerir coloquio al finalizar la cursada a criterio del docente.

Índice

0. Instructivo

Guía de compilación y uso. También, en este mismo documento se encuentran ejemplos de uso en secciones indexadas. El compilador utilizado es **gcc 11.4.0**.

0.1. Compilación

Clonar el repositorio *https://github.com/Hamamura-S/Grupo7Automata/* o descomprimir el .zip, y generar el ejecutable con **`gcc main.c`** `dentro de /src/.`

0.2. Interface

Se mostrará la siguiente interfaz, solicitando teclear 1/2/3 para elegir el proceso

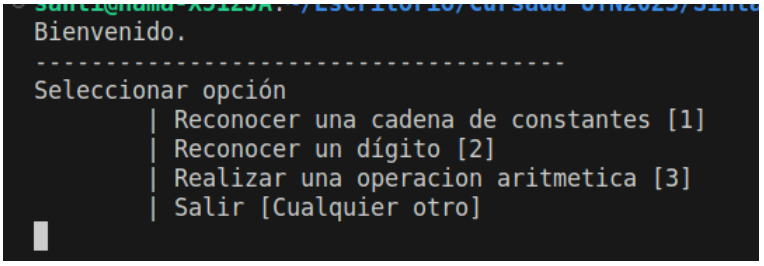

Fig.0.1. Menú inicial

Al finalizar cualquiera de los tres procesos, por ejemplo el 2, se hace una consulta para volver al menú inicial o finalizar.

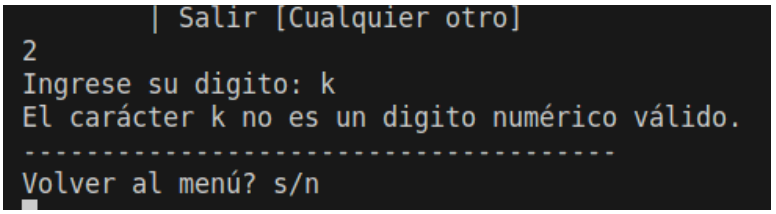

Fig.0.2. Consulta de finalización o reinicio

Aquí, ingresar ‘n’ o cualquier otro carácter tendrá el mismo efecto que ingresar [Cualquier otro] para salir en el menú inicial, terminando el programa.

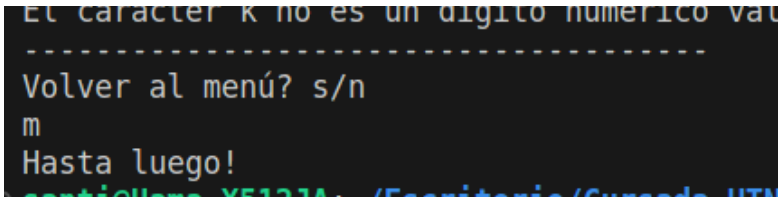

Fig.0.3. Programa finalizado

1. Autómata reconocedor de constantes

Se consideró el siguiente autómata para el reconocimiento de las constantes numéricas solicitado en [1].

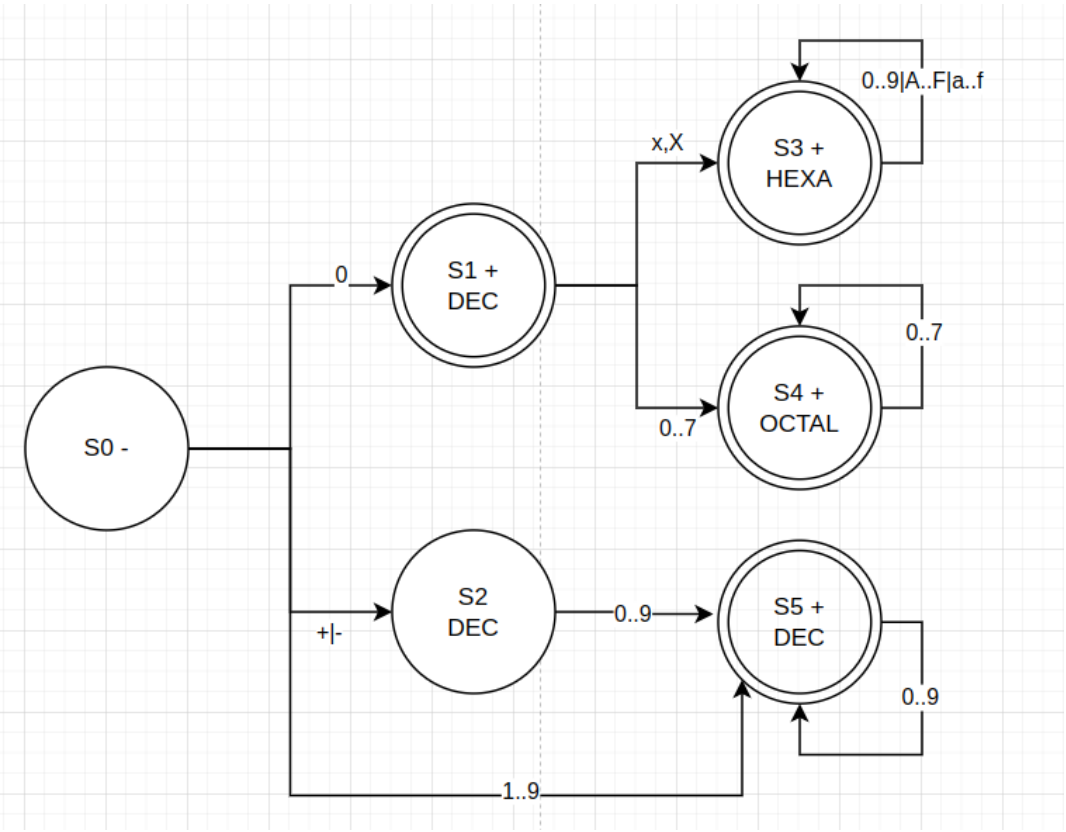

Fig.1. Autómata reconocedor de constantes

Por lo tanto, la tabla de transiciones asociada es esta:

	0	1..7	8, 9	Aa…Ff	X, x	+,-
S0-	**S1**	**S5**	**S5**	R	R	**S2**
S1+	**S4**	**S4**	R	R	**S3**	R
S2	**S5**	**S5**	**S5**	R	R	R
S3+	**S3**	**S3**	**S3**	**S3**	R	R
S4+	**S4**	**S4**	R	R	R	R
S5+	**S5**	**S5**	**S5**	R	R	R
R	R	R	R	R	R	R

1.1. Consideraciones:

1. La palabra “0x” es reconocida como **constante hexadecimal**, cuando en realidad es un **prefijo** y no una constante numérica. Para el ejercicio, se optó por simplificar el autómata en 1 estado adicional en lugar de demostrar más precisión.
2. Los caracteres alfabéticos de los **hexadecimales** (a-f y también la x del prefijo) son reconocidos de forma *case-insensitive.* Es decir, pueden introducirse mayúsculas o minúsculas en la misma constante y será reconocida igualmente.
3. No es un autómata que reconozca la cadena *cruda* separada por “$” que se obtiene por entrada. **Solo reconoce constantes, no expresiones**. El reconocimiento de cada constante separada por “$” se realizó en una lógica que aparte que hace uso de este autómata.
4. La constante **‘0’ es considerada decimal por defecto**. En cambio, ‘00’ se interpreta como un cero octal
5. Los caracteres **‘+’ y ‘-’ caen en un estado no terminal**, haciendo a la constante no válida. Se entiende que por sí solos, no hacen a una constante numérica.

1.2. Diseño

Su programación se encuentra en el archivo “automata.h” y su función de invocación se encuentra en “src/library.h”. El encabezado de esta última es:

```
int reconocerTipo(char palabra[], int len)
```

Donde sus parámetros son un vector de caracteres (palabra) representando la constante a reconocer, y la cantidad de elementos/dígitos en él (len), **sin contar el carácter terminal**. Devuelve 'd': decimal, 'o': octal, 'h': hexadecimal o **-1** en caso de que la palabra no haya sido reconocida.

La anterior se auxilia de la función:

```
int transition(int actualState, char consumed)
```

Desarrollada en “autómata.h”. Dado un estado actual, se encarga de devolver el nuevo estado (int) según el carácter consumido. Realiza un **switch** sobre *actualState*, y en cada **case** se evalúa (if) qué condición cumple *consumed* para seleccionar el nuevo estado. Esta es la función que devuelve -1 en caso de detectar un estado de rechazo.

Para la resolución de [1], se hace uso de `int reconocerTipo(char, int).`

2. Reconocer cadenas de constantes

A partir de aquí, se entiende por “cadena válida”, o “cadena” a un string formado por **constantes numéricas** y el carácter **‘$**’, actuando como separador de cada constante.

La entrada de la cadena por teclado hace uso de la función `fgets`, y la entrada por archivo hace uso de `fgetc`, ambas de la librería `<stdio.h>`. En el caso de la lectura de archivos, se solicitará por consola el nombre del **archivo**, y este **debe encontrarse dentro de la carpeta files**, en el directorio raíz del proyecto. Luego, se leerá carácter a carácter hasta encontrar el EOF.

Nombrando h, d, o, e a los contadores de hexadecimales, decimales, octales y errores respectivamente, y llamando *palabra* al string en el que guardamos cada constante leída, el algoritmo solventa los requerimientos de [1] es:

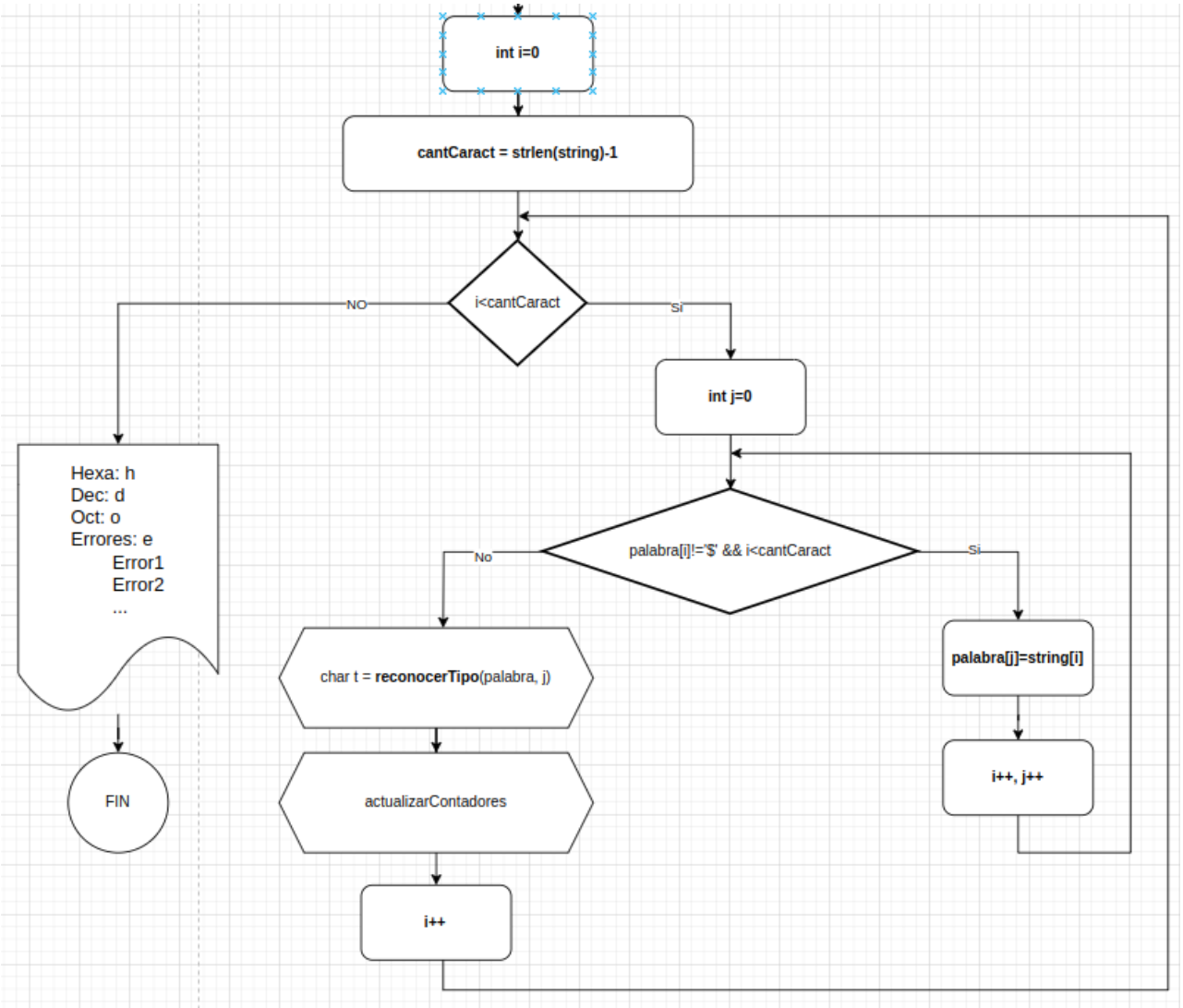

Fig. 2. Algoritmo contador de constantes

2.1. Consideraciones

1. La primera condición es `i<cantCaract` y la segunda es `palabra[i]!='$' && i<cantCaract`. 'j' (jota) es utilizado como contador de caracteres de cada palabra.
2. La cadena no debe contener espacios, estos no están considerados en el alfabeto del autómata y arrojará un error léxico.
3. La cadena puede terminar con $ o no, indiferentemente, pero NO deberá comenzar con $, pues se intentará procesar una palabra vacía.

2.2. Ejemplo, entrada por archivo

El programa solicita 1/2 para decidir la entrada por teclado o archivo.

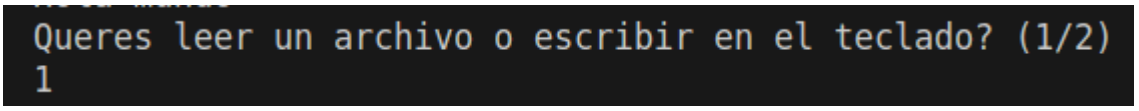

Fig. 3. Usuario requiriendo entrada por archivo

Ingresando una cadena por archivo, solicitará el nombre del archivo que deberá existir en la carpeta "*files/".*

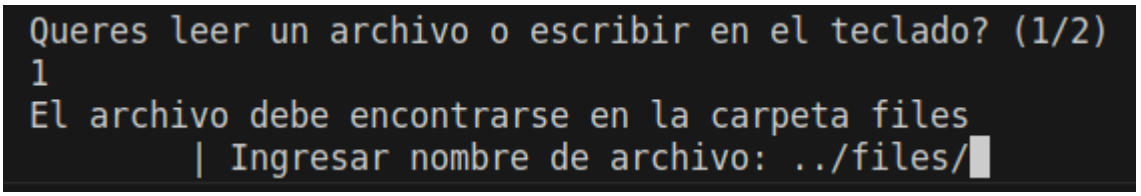

Fig. 4. Sistema solicitando el nombre del archivo

Luego de escribir el nombre del archivo, se imprimirá la cadena leída del mismo, seguido del informe del procesamiento. En este archivo llamado "cadena.txt" se encontraba la cadena *10$0x56f$u87$0710.*

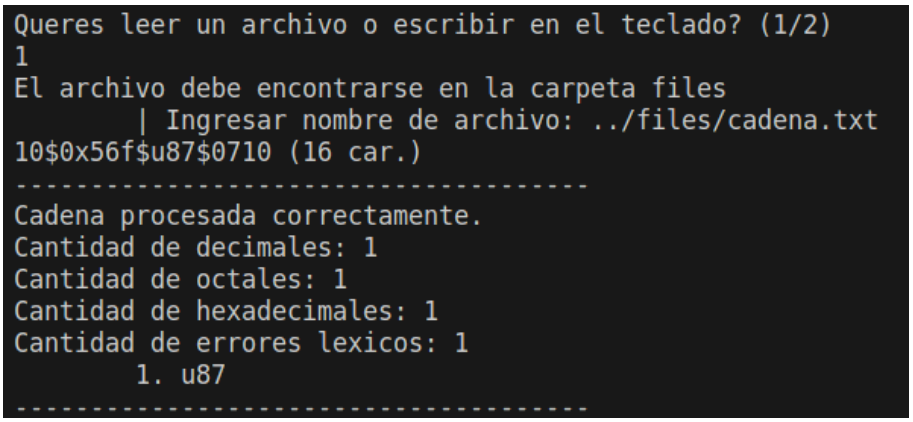

Fig. 5. Impresión de resultados

2.2. Ejemplo, entrada por consola

Se ingresará 2 para acceder a la entrada por consola, y el sistema solicitará el input manual de la cadena a procesar.

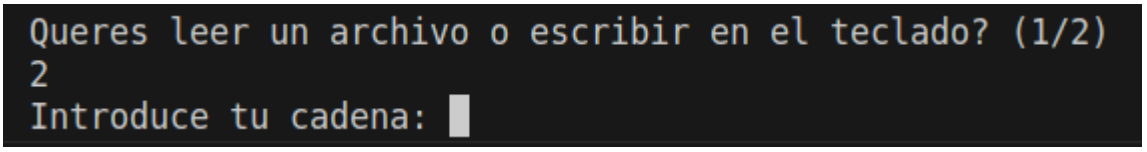

Fig. 6. Usuario requiriendo entrada por teclado

Como ejemplo, se ingresa la cadena 0x123$0$89 conteniendo un hexadecimal y dos decimales.

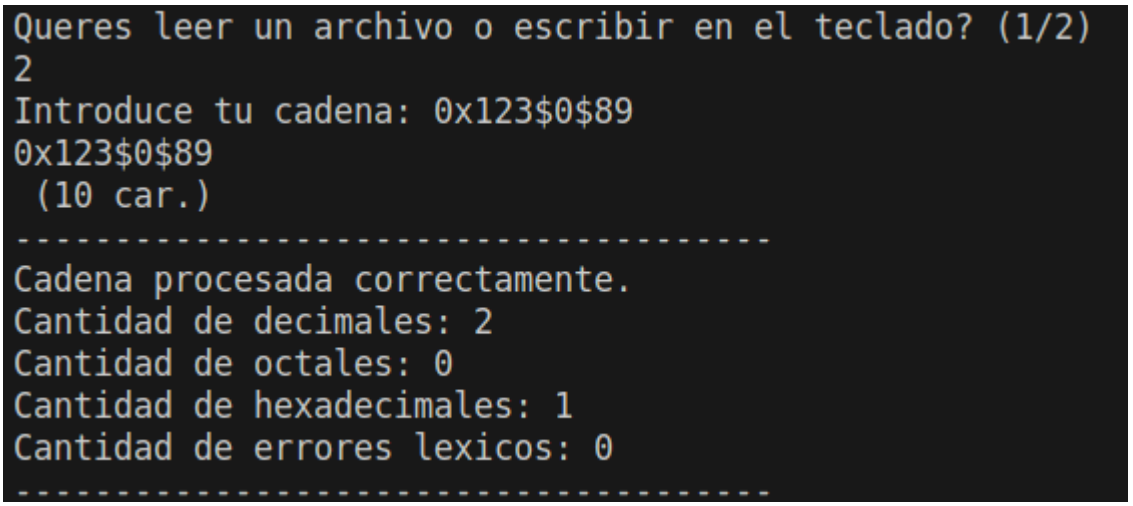

Fig. 7. Impresión de resultados

3. Carácter a dígito entero

La función que satisface los requerimientos de [2] es `int` **`charToInt`**`(char numero)`. Verifica que "`numero`" se encuentre en el rango [48; 57] correspondiente a los valores ASCII de los caracteres '0'-'9'. Caso afirmativo, devuelve el valor entero **numero - '0'** . Caso contrario, devuelve -1.

No realiza entrada/salida de por sí, aunque si es utilizada por [3]. Para funcionar, [2] simplemente chequea el valor de retorno de **charToInt** para imprimir un mensaje u otro.

4. Operaciones aritméticas

Además de las funciones de reconocimiento de constantes desarrollado para [1] y la función conversora de caracteres de [2], se hizo uso de una librería propia desarrollada en el archivo “src/lists.h” donde se implementan funciones de pila y cola como `push`, `pop`, `enqueue` y `dequeue`, bajo un sistema de nodos símplemente enlazados. El análisis léxico se le deriva a la función **reconocerTipo** detallada en la sección 1.2. Se recorre toda la cadena una vez para validar las constantes. **Podría haberse programado un parser con estructura de árbol**, se optó por la alternativa de listas para simplificar el ejercicio.

4.1. Precedencia

Recibida y validada la expresión algebráica, se realiza una lectura de izquierda a derecha, almacenando en una nueva cadena los operandos y operadores, también ordenando las operaciones por orden de precedencia descendente. Se optó por modificar la notación de la expresión de infija (default) a postfija (primero operandos y luego operadores). De esta manera la cadena se vuelve más tratable en una estrategia de pilas.

$$(infix\ natural)\ 100\ +\ 2\ *\ 5\ -\ 5\ \ ->\ (postfix\ reordenado)\ 100\ 2\ 5\ \ *\ \ +\ \ 5\ \ -$$

La función que realiza este proceso es **`char *infixToPostfix(char *expresion)`**, desarrollada en infix.h. Al tener la expresión ordenada por la precedencia de los operadores, solo resta evaluarla secuencialmente.

4.2. Evaluación

La función double `evaluarPostfijo(char *expresion)` evalúa la expresión devuelta por `infixToPostfix(char)` y devuelve el valor que se imprimirá como resultado. Hace uso de `charToInt(char)` para convertir cada carácter leído a un entero, y luego sumarizar sus productos por potencias de 10 para formar los operandos numéricos. Asume que la validación de las constantes se ha hecho anteriormente.

Cada operando leído se guarda en una pila, y al detectar un operador se remueven (pop) los últimos dos operandos guardados, se operan y se almacenan de vuelta en la pila. El proceso se repite hasta que el último valor almacenado es el resultado buscado. Se lo devuelve y finaliza el proceso.

4.3. Ejemplo

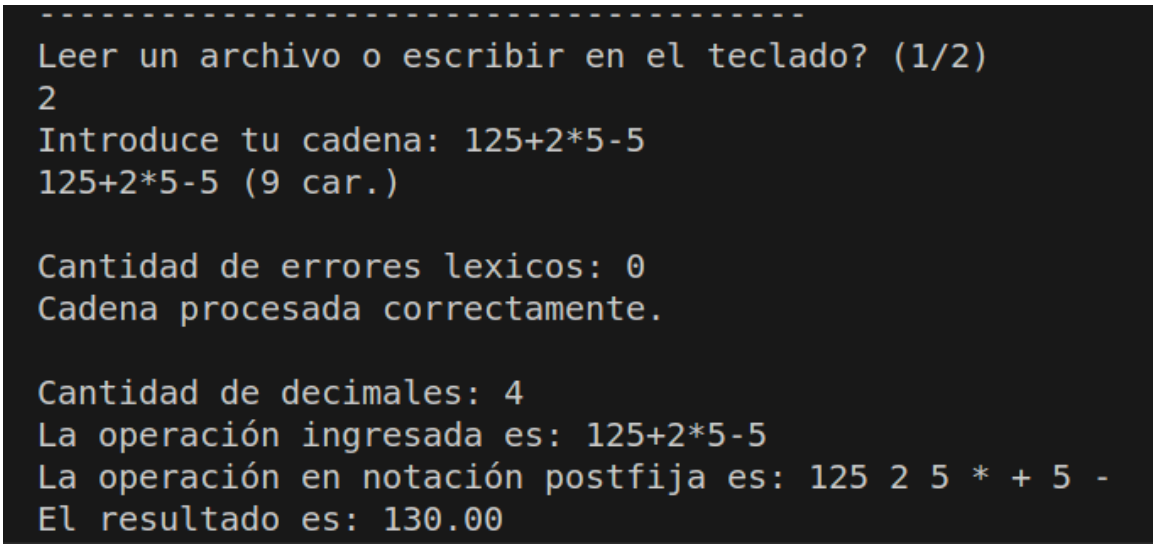

Fig.8. Invocación de rutina de operaciones aritméticas.

4.4. Consideraciones

1. El resultado es de tipo float, pero todos los operandos deben ser enteros.
2. No se aceptan hexadecimales ni octales
3. En caso de errores léxicos, se muestran y se vuelve al menú principal.
4. No puede operar con paréntesis.