Práctica Final. Sistemas de ecuaciones lineales

En el trabajo final, se deberán implementar una serie de funciones que nos permitirán resolver sistemas de ecuaciones lineales con dos incógnitas. Es decir, sistemas como este:

$$\begin{aligned} ax + by &= c \\ a'x + b'y &= c' \end{aligned}$$

La resolución matemática del sistema será realizada por una función proporcionada por una biblioteca externa en `Tablon`. El grupo deberá implementar una función de entrada para parsear una ecuación, mas una función de salida para generar un string con la solución. Por último, se creará una tercera función que integrará las funciones de entrada/salida con la función de resolución matemática, de tal forma que, a partir de dos cadenas de entrada que forman un sistema de ecuaciones con dos incógnitas, devuelva un string con la solución del sistema. Para poder integrar las tres funciones y que se evalúen en el `Tablon`, estas deberán cumplir con la especificación establecida y trabajar con los tipos de datos definidos a continuación.

1. Tipos de datos

Las funciones, además de strings, trabajarán con otros dos tipos de datos como parámetros de entrada/salida. Se necesitará un tipo de datos para representar una ecuación lineal y un segundo tipo de datos para representar la solución de un sistema de ecuaciones. Un sistema de ecuaciones sería simplemente un conjunto de 2 ecuaciones lineales, por lo que no será necesario crear un nuevo tipo para representar sistema de ecuaciones.

1.1. Tipo *Ecuacion*

Este tipo de datos contiene tres enteros para representar los dos coeficientes y el término independiente. Además contiene 2 variables extra para guardar el nombre de las incógnitas, ya que no tienen porque ser forzosamente x e y. Cualquier carácter alfabético sería válido. Nótese que aunque para representar un carácter nos bastaría el tipo `char`, para facilitar la gestión de la memoria los caracteres se representarán usando tipos `int`.

El código `C` para representar el tipo *Ecuacion* sería el siguiente:

```
/*Ecuación lineal de la forma ax +by = c */
struct Ecuacion{
        int a;  //coeficiente x
        int b;  //coeficiente y
        int c;  //término independiente
        int name_incognita1; //Nombres de las incógnitas
        int name_incognita2; //Cualquier carácter del alfabeto es válido
};
```

Por ejemplo, la ecuación $14x - 3y = 4$ se representaría como:

```
struct Ecuacion ec1 = {14, -3, 4, 'x', 'y'};
```

Mientras que la ecuación $-25z + q = -15$ se representaría como:

```
struct Ecuacion ec2 = {-25, 1, -15, 'z', 'q'};
```

1.2. Tipo *Solucion*

Este tipo de datos contiene un total de 7 enteros. Aunque el tipo *Ecuacion* solo admite coeficientes enteros (es decir, $a, b, c \in \mathbb{Z}$), la solución de un sistema puede tener soluciones racionales ($x, y \in \mathbb{Q}$). Para evitar usar números en coma flotante, los números racionales se representarán vía software. Por ello, necesitamos 3 enteros por incógnita: el primero indica la parte entera, el segundo indica el desplazamiento de la parte fraccionaria respecto a la coma decimal, y el tercero indica la parte fraccionaria. Por ejemplo, al representar el número $104,0061$, su parte entera sería 104, su desplazamiento sería 2 y su parte fraccionaria sería 61. $213{,}35$ se representaría con los valores $\{231, 0, 35\}$. Si el número no tiene parte fraccionaria, tanto el desplazamiento como la parte fraccionaria

son cero. Por último, un último entero nos indica el tipo de solución obtenida: compatible determinado, compatible indeterminado o incompatible. Si la solución del sistema es indeterminada o incompatible, el valor del x e y es indiferente y se puede ignorar.

El código `C` para representar el tipo *Solucion* sería el siguiente:

```
/*Solución al sistema de ecuaciones.
  Para evitar usar instrucciones en punto flotante,
  los números racionales se representan vía software*/
struct Solucion{                    /*Tipo de solución:*/
        int tipo;                   /*0 --> Compatible determinado
                                    1 --> Compatible indeterminado
                                    2 --> Incompatible*/
        int x_entero;               //primera incógnita
        int x_desp;
        int x_decimal;
        int y_entero;               //segunda incógnita
        int y_desp;
        int y_decimal;
        int x_name; //Nombres de las incógnitas
        int y_name; //Cualquier carácter del alfabeto es válido

};
```

Por ejemplo, la solución $x = 45{,}23$ $y = -15{,}0001$ se representaría como:

```
struct Solucion sol1 = {0, 45, 0, 23, -15,3,1, 'x','y'};
```

Mientras que , la solución $t = -100$ $s = 2{,}5$ se representaría como:

```
struct Solucion sol2 = {0, -100, 0, 0, 2,0,5, 't','s'};
```

Este formato de punto flotante software tiene un problema: no nos permite representar números como `-0.5` o `-0.001`. Esto es debido a los enteros en complemento a 2 tiene una única representación del cero, a diferencia de los enteros en complemento a 1. Para solventar este problema, utilizaremos el mínimo del rango de los enteros (`0x80000000` o `-2147483648`) para representar el `-0`. Así, la solución $x = 0{,}5$ $y = 0{,}5$ se representaría como:

```
struct Solucion sol3 = {0, 0, 0, 5, 0, 0, 5 'x','y'};
```

Mientras que la solución $x = -0{,}5$ $y = 0{,}5$ se representaría como:

```
struct Solucion sol4 = {0, -2147483648, 0, 5, 0, 0, 5 'x','y'};
```

2. Funciones

A continuación, se describen la especificación de las funciones y su comportamiento esperado. Esta especificación debe servir para el alumno como un "contrato": cualquier otra funcionalidad o restricción no descrita en las siguientes secciones no será evaluada. En la documentación de cada función también se incluye como utilizar los registros de argumento ($\$a_i$) y de retorno o ($\$v_i$) para cada función. Utilizar los registros de otra forma provocará que el programa no se pueda testear en la cola `lb_sistemas` (consultar sección 3.1) en los casos de las funciones *String2Ecuacion* y *Solucion2String*, en las funciones *Cramer* y *ResuelveSistema*, provocará que el programa no funcione correctamente.

2.1. Función *String2Ecuacion*

Esta función recibe una cadena de caracteres y produce como resultado un objeto del tipo *Ecuacion*. Si la cadena puede convertirse en una ecuación lineal, la función devuelve 0. En caso contrario, devuelve un código indicando el fallo ocurrido. Para su implementación, se recomienda la reutilización de la función `atoi`, aunque es posible que requiera de ligeras modificaciones para adaptarse a la especificación. El código `C` de la especificación de *String2Ecuacion* es el siguiente:

```
/* Parámetros: $a0 <- input_cad; $a1 <- output_ec
   Retorno:      $v0 <- salida de error
   Códigos de error:   0: Ecuación correcta
                       1: Sintasix incorrecta
                       2: Overflow en término o coeficiente
                       3: Overflow al reducir términos o coeficientes
                       4: Número de incógnitas incorrectas   */

int String2Ecuacion(char* input_cad, Ecuacion* output_ec);
```

Las cadenas debe cumplir las siguientes condiciones para poder considerarse correctas y ser transformadas a un objeto *Ecuacion*:

- La cadena de caracteres debe cumplir la siguiente sintaxis para considerarse una ecuación válida. De no cumplirse alguna de estas condiciones, *String2Ecuacion* devuelve un error 1 (sintaxis incorrecta).
  - Una ecuación estará compuesta por una serie de 1 a n de términos, seguidos de un carácter $=$, seguidos de 1 a n' términos. Es decir:
    - "$10x + y = 1$" o "$5x + y - 4 = 45y - 34x + 100$" son ecuaciones válidas.
    - "$x + y + 3$" o "$x + y =$" son ecuaciones inválidas.
  - Los términos deben cumplir este formato:
    - Una serie de espacios opcionales antes de comenzar el término.
    - El carácter $+$ o $-$. Este carácter es opcional en el primer término de cualquiera de las dos partes de la ecuación, pero es obligatorio en el segundo y siguientes. Es decir, tanto "$+3x + 4y = 0$" como "$3x + 4y = 0$" son correctos, pero "$3x\ \ 4y = 0$" no.
    - Una serie de dígitos numéricos que representan los coeficientes y términos independientes. Opcionales si el coeficiente es 1 o -1. Es decir, los términos x y $1x$ son válidos y equivalentes.
    - Un carácter alfabético que representa una incógnita, opcional para poder representar términos independientes.
    - Ejemplos de términos válidos aplicando estas reglas: $+5$, -5, $+123x$, $+123X$, $-5678t$, $+q$, $-q$; si además el término es el primero en una de las dos partes de la ecuación: 5, $123x$, $123X$, q.
    - Ejemplos de términos incorrectos: $+$, $++23$, $+-23$, $+\ \ 23$, $23\ \ x$, $-23\,\%$, $-23\$$, $-23abc$
- Los coeficientes y términos independientes de la ecuación deben poder codificarse en enteros de 32 bits en complemento a 2. Si aparece desbordamiento durante la lectura de un término o coeficiente, la cadena no puede convertirse a *Ecuacion* y *String2Ecuacion* devuelve un error 2 (overflow término/coeficiente).
- Como la ecuación puede tener múltiples términos, deben reducirse para representarlos en el objeto *Ecuacion*. Por ejemplo, "$x - y = 2y + 5x + 5$" sería reducida a la ecuación equivalente "$-4x - 3y = 5$". Si aparece desbordamiento durante la reducción de términos o coeficientes, la cadena no puede convertirse a *Ecuacion* y *String2Ecuacion* devuelve un error 3 (overflow reducción).
- Una ecuación debe contener una o dos incógnitas. En otro caso, la cadena no puede convertirse a *Ecuacion* y *String2Ecuacion* devuelve un error 4 (número de incógnitas incorrectas).
  - "$2x = 3$" y "$10x + y = 1$" son ecuaciones válidas.
  - "$x + y + z = 3$" o "$5 = 3$" son ecuaciones inválidas.
- Consideramos como primera incógnita a la primera en aparecer en la cadena.

2.2. Función *Cramer*

Esta función recibe dos objetos tipo *Ecuacion* que forman un sistema de ecuaciones lineal, lo resuelve utilizando la regla de *Cramer*, produciendo como resultado un objeto *Solución* y siempre retorna una solución, ya sea un sistema de ecuaciones compatible o no, por lo que no devuelve ningún código de error. **Esta función no tiene que ser implementada por los alumnos**. El código `C` de la especificación de *Cramer* es el siguiente:

```
/* Parámetros:
        $a0 <-- ec1
        $a1 <-- ec2
        $a2 <-- sol
   Siempre resuelve el sistema, por lo que no es necesario retornar
      salida de error
*/
void Cramer(Ecuacion* ec1, Ecuacion* ec2, Solucion* sol);
```

2.3. Función *Solucion2String*

Esta función recibe un objeto tipo *Solucion*, que contiene la solución de un sistema de ecuaciones lineal, y genera una cadena de caracteres con la información del objeto. Esta función siempre retorna una cadena, ya sea la solución de sistema de ecuaciones compatible o no, por lo que no devuelve ningún código de error. Para su implementación, se recomienda reutilizar la función `itoa`. El código `C` de la especificación de *Solucion2String* es el siguiente:

```
/* Parámetros:
        $a0 <-- input_sol
        $a1 <-- output_string
   Siempre genera una cadena, por lo que no es necesario retornar error
*/
void Solucion2String(Solucion* input_sol, char* output_string);
```

La cadena generada debe cumplir las siguientes condiciones, dependiendo del campo `input_sol->tipo`:

0: Solución a un sistema compatible determinado. Devuelve una cadena de la forma:

`x=245.34 y=-123`

teniendo en cuenta que:

- `x` es el carácter contenido en `input_sol->x_name`.
- `y` es el carácter contenido en `input_sol->y_name`.
- `245.34` es el número en punto flotante codificado en `input_sol->x_entero`, `input_sol->x_desp` e `input_sol->x_decimal`,
- `-123` es el número en punto flotante codificado en `input_sol->y_entero`, `input_sol->y_desp` e `input_sol->y_decimal`.
- Si la parte fraccionaria del número en punto flotante es cero, no se incluye al generar la cadena.
- El valor de ambas incógnitas está separado por el carácter espacio.

1: Solución a un sistema compatible indeterminado. Devuelve la cadena "`INDETERMINADO`".

2: Solución a un sistema incompatible. Devuelve la cadena "`INCOMPATIBLE`".

2.4. Función *ResuelveSistema*

Esta función recibe dos cadenas de caracteres, que forman un sistema de ecuaciones, y produce como resultado una cadena de caracteres que contiene la solución del sistema. Para ello se utilizarán las tres funciones explicadas anteriormente. Si las cadenas forman un sistema de ecuaciones válido, la función devuelve 0. En caso contrario, devuelve un código indicando el fallo ocurrido. Los fallos solo pueden ocurrir por dos motivos:

- Una de las dos cadenas de entrada no puede ser convertida a ecuación con dos incógnitas por la función *String2Ecuacion* (consultar Sección 2.1). En ese caso, *ResuelveSistema* devuelve el mismo código de error devuelto por *String2Ecuacion*.
- El sistema de ecuaciones tiene 3 o más incógnitas. Por ejemplo, consideremos el siguiente sistema:

$$\begin{aligned} 3x + y &= 2 \\ x - z &= 0 \end{aligned}$$

  Ambas cadenas podrían ser convertidas a *Ecuacion*. Pero el sistema tiene tres incógnitas (x, y y z), por lo que *ResuelveSistema* devuelve el código de error 5 (sistema de ecuaciones no válido). En cambio, el sistema:

$$\begin{aligned} 3x + y &= 2 \\ y - x &= 0 \end{aligned}$$

  sí se puede resolver. Aunque las incógnitas aparecen en diferente orden, solo hay dos, x e y. En este caso, se debería intercambiar los campos de la primera y la segunda incógnita de la segunda ecuación para que la función *Cramer* produzca la solución correcta.

En cualquier otro caso, el sistema podrá resolverse mediante la función *Cramer* y obtener una cadena de caracteres con la solución utilizando la función *Solucion2String*. El código `C` de la especificación de *ResuelveSistema* es el siguiente:

```
/* Parámetros: $a0 <-- input_ec1; $a1 <-- input_ec2; $a2 <-- output_sol
   Retorno:    $v0 <-- salida de error
   Códigos de error:   0: Sistema de ecuaciones resuelto
                       1: Sintasix incorrecta
                       2: Overflow en término o coeficiente
                       3: Overflow al reducir términos o coeficientes
                       4: Número de incógnitas incorrectas
                       5: Sistema de ecuaciones no válido */

int ResuelveSistema(char* input_ec1, char* input_ec2, char* output_sol);
```

3. Evaluación

La práctica será evaluada con la herramienta `Tablon`, disponible en `http://tablon-aoc.infor.uva.es/`. Para poder evaluar la práctica, el fichero `.asm` enviado al `Tablon` debe incluir las funciones *String2Ecuacion*, *Solucion2String* y *ResuelveSistema*.

Una vez enviado el código, `Tablon` realizará una serie pruebas, testeando diferentes valores de entrada. Una vez finalizada la batería de pruebas, se podrá consultar el `Tablon` el número de pruebas superadas y el porcentaje de pruebas completado, además de otra información relevante, como el número de instrucciones ejecutadas, tiempo de ejecución, CPI y el número de excepciones en tiempo de ejecución. Estas métricas sirven para elaborar el ranking y desempatar en caso de empate en el número de casos de prueba superados. Para calcular el tiempo de ejecución en número de ciclos, el programa se ejecutará en un procesador segmentado de 5 etapas con anticipación completa, saltos resueltos en la etapa ID y predictor de salto no tomado.

A diferencia de las prácticas guiadas, **la calificación del trabajo final dependerá del número pruebas superadas, el tiempo de ejecución y el número de excepciones**. Aumentar el número de casos de prueba superados y reducir el tiempo de ejecución aumentará la puntuación, y por tanto, la calificación, mientras que aumentar el número de excepciones mermará la puntuación y la calificación. El resto de métricas no serán evaluadas.

Los criterios de calificación del programa `Tablon` se publicarán cuando comience el concurso, ya que esté no comenzará al comienzo del trabajo final, sino hacia la mitad del tiempo disponible para su desarrollo. Y como ocurría en prácticas anteriores, la calificación no depende exclusivamente del resultado del `Tablon` y el código se revisará posteriormente de forma manual.

Para poder comprobar el avance de los grupos, cada grupo debe entregar semanalmente un fichero `.asm` con los avances realizados (aunque todavía haya funciones no implementadas). Al finalizar el concurso, se debe entregar

en el Aula Virtual el fichero `.asm` que obtuviera el mejor resultado en el ranking de la herramienta `Tablon`. El nombre del fichero para el Aula Virtual será el número de *request* del `Tablon` con 5 cifras y la extensión `.asm`

La cola para evaluar el trabajo en `Tablon` es `lb_sistemas`.

`python ./client -u` *usuario* `-x` *password* `-q lb_sistemas` *NombrePrograma*

Para mandar pruebas y testear las tres funciones, se puede emplear la cola `mars_sistemas`.

`python ./client -u` *usuario* `-x` *password* `-q mars_sistemas` *NombrePrograma* `--` "`str`"
donde *str* es una cadena, delimitada por comillas dobles, con el argumento a testear.

Para más información sobre el lanzamiento de pruebas individuales, consultar la Sección 3.1.

La cola `lb_sistemas` se habilitará cuando comience el concurso. La cola `mars_sistemas` estará habilitada desde el inicio del trabajo final.

Se exigirán entregas semanales de lo realizado cada semana aunque, evidentemente, esté incompleto o no funcione.

Fecha límite de la primera entrega semanal en el Aula Virtual: 7 de noviembre a las 23:55 h.

Fecha límite de la entrega final en el Aula Virtual: 3 de diciembre a las 23:55 h.

No se evaluarán las entregas del `Tablon` que no se hayan depositado dentro del plazo en el Aula Virtual.

3.1. Uso de cola de mars_sistemas

La cola `mars_sistemas` permite ejecutar pruebas individuales de las 3 funciones que los alumnos deben implementar. Como las funciones tienen varios parámetros de entrada y estructuras con múltiples campos, tendrán que ser especificados a través del argumento `str`. Además, para que el fichero `.asm` compile, debe incluir las 3 funciones. Por ello, se recomienda comenzar con un fichero que contenga 3 funciones vacías:

```
String2Ecuacion:
                jr $ra

Solucion2String:
                jr $ra

ResuelveSistema:
                jr $ra
```

Este fichero compilará y ejecutará la prueba deseada, aunque no produzca ningún resultado. También nos permite hacer pruebas con una función, aunque el resto no estén implementadas.

En cuanto a la cadena que tenemos que pasar como argumento, debe ser una única cadena, delimitada por comillas dobles. Como necesitaremos múltiples valores para evaluar las funciones, pero solo podemos incluir un argumento, el carácter # servirá de delimitador entre campos, siendo el primer campo un entero entre 0 y 2 que nos indicará que función testear. En caso de indicar un entero fuera de este rango, se nos mostrará el mensaje `ERROR OPCION USO COLA mars_sistemas`. A continuación se detalla como realizar pruebas con las tres funciones.

String2Ecuacion

El formato del argumento `str` será el siguiente:

`0#cadena_a_transformar`

Si la función termina sin errores, la salida de `Tablón` serán los campos del objeto *Ecuación* generados. En caso contrario, mostrará la palabra `ERROR` seguida del código de error e información sobre el mismo.

Por ejemplo:

- Argumento: "`0#2x +4y = -5`". La salida correcta de `Tablón` es : `{2, 4, -5, x, y}`
- Argumento: "`0#2x +4y -5`". La salida correcta de `Tablón` es : `ERROR 1 - SINTAXIS`

Solucion2String

El formato del argumento `str` será el siguiente:

```
1#tipo#x_entero#x_desp#x_decimal#y_entero#y_desp#y_decimal#x_name#y_name
```

La salida de `Tablón` serán el string generado por la función *Solucion2String*. Por ejemplo:

- Argumento: "`1#0#1#0#5#5#0#0#q#t`". La salida correcta de `Tablón` es: `q=1.5 t=5`
- Argumento: "`1#0#2#2#55#7#0#1#z#x`". La salida correcta de `Tablón` es: `z=2.0055 x=7.1`
- Argumento: "`1#1#2#2#55#7#0#1#z#x`". La salida correcta de `Tablón` es: `INDETERMINADO`
- Argumento: "`1#2#2#2#55#7#0#1#z#x`". La salida correcta de `Tablón` es: `INCOMPATIBLE`

ResuelveSistema

El formato del argumento `str` será el siguiente:

```
2#ecuacion1#ecuacion2
```

Si la función termina sin errores, la salida de `Tablón` será el string con la solución. En caso contrario, mostrará la palabra `ERROR` seguida del código de error e información sobre el mismo.

Por ejemplo:

- Argumento: "`2#t-s=0#2t+s=3`". La salida correcta de `Tablón` es : `t=1 s=1`.
- Argumento: "`2#t+z=0#2t+s=3`". La salida correcta de `Tablón` es : `ERROR 5 - SISTEMA INVALIDO`