Ejercicios prácticos sobre Programación en C

Entrega 1

Programación Para Sistemas (PPS) 2022/23

Ejercicio 1.

Escribe un programa, ***invertir.c***, que lea enteros del canal de entrada estándar y los almacene en un array para imprimirlos en la salida estándar en orden inverso. Tu programa admitirá un máximo de M enteros (defina M como 1000). En el momento en el que haya algo diferente a un entero o se llegue a M números, el programar invertirá todos los enteros leídos hasta ese momento y empezará a leer la siguiente secuencia. Se supone que el final de cada secuencia se marca con una única palabra. Las secuencias en la entrada estándar no están organizadas mediante líneas, se debe procesar independientemente de la posición de los saltos de línea. Al escribir, se debe escribir cada secuencia de números en la misma línea de texto. Cada número debe separarse del siguiente exactamente por un **único espacio** en blanco. El último número debe seguirse directamente del salto de línea. **No** debe existir un espacio en blanco antes del salto de línea. Tampoco debe existir ningún espacio en blanco antes del primer número.

Para no tener que teclear números desde el teclado os sugerimos que pongáis los números en un fichero, digamos *numeros.txt*:

```
1 8 4000 2 -23 final 51 87 fin -4
-3 2 terminar
```

Si suponemos que has compilado tu programa a **`invertir`**, esperamos que la orden: `./invertir < numeros.txt` produzca esta salida:

```
-23 2 4000 8 1
87 51
2 -3 -4
```

Ejercicio 2.

Escribe un programa, ***`multiplicar.c`***, que lea de la entrada estándar dos matrices de números enteros y las multiplique. Puedes asumir que las matrices están codificadas de la siguiente forma en la entrada estándar:

- Un entero positivo `m` que indica el número de filas de la primera matriz.
- Un entero positivo `n` que indica el número de columnas de la primera matriz (que coincide con el número de filas de la segunda).
- Un entero positivo `p` que indica el número de columnas de la segunda matriz.
- *m x n* enteros: los n primeros son la primera fila de la primera matriz, los n segundos la segunda fila, etc.

- *n x p* enteros: los p primeros son la primera fila de la segunda matriz, los p segundos la segunda fila, etc.

Se puede suponer que `m`, `n` y `p` están **entre 1 y 64**. El resultado serán *m x p* enteros donde los `p` primeros son la primera fila de la matriz multiplicación, los `p` segundos la segunda fila, etc.

Para no tener que teclear las matrices desde el teclado os sugerimos que las pongáis en un fichero, digamos *matrices.txt*:

```
3

2

3

1 2

-1 0

-3 -1

2 0 1

-5 2 3
```

Si suponemos que has compilado tu programa a **`multiplicar`**, esperamos que la orden*: `./multiplicar < matrices.txt`* produzca esta salida.

```
-8 4 7

-2 0 -1

-1 -2 -6
```

Para escribir la matriz resultado se debe escribir una fila por línea, cada número debe separarse del siguiente exactamente por un **único espacio** en blanco. El último número debe seguirse directamente del salto de línea. **No** debe existir un espacio en blanco antes del salto de línea. Tampoco debe existir ningún espacio en blanco antes del primer número.

Ejercicio 3.

La ecuación dada por a * x^2 + b * x + c = 0, más conocida por *ecuación de segundo grado* tiene una solución bien conocida a través de una fórmula que permite obtener las raíces reales de la ecuación cuando se cumplen algunas condiciones. Sin embargo, es posible dar la solución para cualesquiera valores de a, b y c, excepto, naturalmente, que tanto a como b sean cero (sea cual sea el valor de c). Se pide:

Escribir un archivo **`ec2g.c`** con la implementación de una función cuya cabecera es:

```
int resolver(double a, double b, double c, double* px1, double *px2);
```

que resuelva la ecuación de segundo grado de acuerdo a las siguientes especificaciones:

1) Si las soluciones son reales se deben guardar en las variables apuntadas por `px1` y `px2`.

2) Si las soluciones son imaginarias se debe guardar la parte real de la solución en (la variable apuntada por) `px1` y la parte imaginaria en (la variable apuntada por) `px2`. Como las dos soluciones imaginarias son conjugadas, es suficiente con guardar una solución, se debe guardar aquella con **parte imaginaria mayor que cero**.

3) Si la solución no existe se deben fijar las variables apuntadas por `px1` y `px2` a cero.

4) Si existe una única solución (ecuación de primer grado), esta se debe guardar en `px1` y debe guardarse el valor *NaN* en `px2`. Para generar NaN (*Not a Number*) se debe calcular la raíz cuadrada de –1. Es decir: **`sqrt(-1.0)`**.

5) La función debe devolver el número entero correspondiente al caso encontrado. Por ejemplo, si la solución no existe la función debe devolver 3.

Para comprobar la función se debe escribir el programa principal en un archivo: **`mainEc2g.c`** y compilar con el archivo anterior para obtener el archivo ejecutable **`mainEc2g`**, el programa principal debe leer los valores de a, b y c del canal de entrada estándar y escribir en el canal de salida el entero retornado por la función junto con los valores obtenidos para x1 y x2.

Para no tener que escribir los datos desde el teclado os sugerimos que las pongáis en un fichero, digamos *ecuaciones.txt*:

```
1.0   -2.0  1.0
-1.0   -2.0  1.0
3.0   -2.0  -8.0
-3.0   -2.0  -8.0
2.0   3.0   5.0
0.0   0.0   4.0
2.0   0.0   6.0
0.0   4.0   11.0
```

Si suponemos que has compilado tu programa a un archivo ejecutable **`mainEc2g`**, esperamos que la orden: `./ mainEc2g < ecuaciones.txt` produzca esta salida (es indiferente el signo que pueda tener el cero ó nan; el formato puede ser distinto):

```
Caso 1: 1.000000, 1.000000
Caso 1: -2.414214, 0.414214
Caso 1: 2.000000, -1.333333
Caso 2: -0.333333, 1.598611
Caso 2: -0.750000, 1.391941
Caso 3: 0.000000, 0.000000
Caso 2: -0.000000, 1.732051
```

```
Caso 4: -2.750000, -nan
```

Opciones de compilación.

Todos los programas deben compilarse con las opciones `-ansi -pedantic -Wall -Wextra -Werror`. Además, para compilar el ejercicio 3 es necesario añadir la opción `-lm` al final del comando de compilación.