Décimo Concurso Local de Programación ACM ICPC

Universidad Autónoma Metropolitana Unidad Azcapotzalco

Examen final, 27 de agosto a 2 de septiembre de 2013

Inicio: 10:00 del 27 de agosto. Fin: 23:00 del 2 de septiembre.

Instrucciones: A continuación encontrarás los enunciados de tres problemas. Cada problema tiene un valor de 100 puntos, los cuales se obtendrán de las salidas que tu programa entregue para cada una de 10 entradas distintas. Todos los programas se evaluarán en Linux. El que tu programa funcione con el ejemplo no quiere decir que funcionará siempre. Ningún programa se ejecutará por más de 5 segundos.

Para cada problema que resuelvas deberás enviar por correo el código fuente de un programa a la dirección uam13acm@gmail.com indicando claramente tu nombre completo. De preferencia envía todos tus códigos en un solo correo. El nombre de los archivos que envíes debe ser de la forma xxx.zzz, donde xxx es el nombre del problema y zzz es la extensión, según el lenguaje que hayas usado. Si así lo deseas, puedes resolver cada problema en un lenguaje distinto.

Tu programa deberá leer y escribir exactamente los datos que se indican (ni más ni menos) usando los **archivos** de entrada y salida indicados. En particular, tu programa no deberá: borrar la pantalla, escribir ningún tipo de letrero adicional, esperar ninguna tecla, usar la biblioteca conio, etc.

Quienes programen en C deberán usar el gcc, quienes programen en C++ deberán usar el g++ y quienes programen en Java deberán usar el gcj de la siguiente manera:

```
gcc xxx.c -lm -o xxx
g++ xxx.cpp -lm -o xxx
gcj xxx.java –main=xxx -Wall -o xxx
```

Nota: Todos los puntos calculados que resulten fraccionarios se redondearán siempre hacia abajo

Problema 1: Acomodando cuadritos de colores (100 puntos)

Código fuente: acc.c, acc.cpp, acc.java

Archivo de entrada: acc.ent ***Archivo de salida: acc.sal***

Imagina que tienes una cuadrícula de **N** por **N** y que en las $\mathbf{N}^2$ entradas de la cuadrícula hay **N** cuadritos de cada uno de **N** colores. El objetivo es colocar todos los cuadritos del mismo color en la misma columna en un cierto orden. Para ello, puedes llevar a cabo una secuencia de **M** operaciones de dos tipos: intercambiar dos cuadritos que son adyacentes horizontalmente o intercambiar dos cuadritos que son adyacentes verticalmente. Por simplicidad, los colores los denotaremos por los enteros del 1 al **N** y deben quedar en el orden del 1 al **N** de izquierda a derecha.

El ejemplo de la entrada se puede resolver con **M** = 3 intercambios. Primero intercambia el 3 y el 2 que están en el primer renglón. Después intercambia el 1 y el 2 de la segunda columna. Finalmente intercambia el 2 y el 1 del segundo renglón. Estos tres movimientos los denotaremos por las coordenadas de los cuadritos intercambiados. Por ejemplo, el primer intercambio se representa por las coordenadas (1, 2) y (1, 3), donde primero se anotó el renglón y después la columna.

Escribe un programa que encuentre una secuencia de intercambios que acomode los cuadritos como se pide y que sea tan corta como te sea posible.

Entrada: Un entero **N**, seguido de una matriz de **N** por **N** enteros. Puedes suponer que $2 \leq \mathbf{N} \leq 10$ y que en la matriz hay exactamente **N** enteros de cada uno de los valores del 1 al **N**.

Salida: Un entero **M**, seguido de **M** parejas de coordenadas correspondientes a los intercambios.

Evaluación: En caso de que la secuencia sea válida, 2+8**C**/**M** puntos, donde **C** es la cantidad mínima conocida de operaciones con las que se sepa que se puede resolver el caso de entrada.

Ejemplo de entrada	*Ejemplo de salida*
3 1 3 2 2 2 3 1 1 3	3 1 2 1 3 2 2 3 2 2 1 2 2

Problema 2: Completando las contraseñas del celular (100 puntos)

Código fuente: ccc.c, ccc.cpp, ccc.java

Archivo de entrada: ccc.ent ***Archivo de salida: ccc.sal***

Generalmente, los teléfonos celulares tienen programas que bloquean las funcionalidades del equipo después de que éste no se utiliza durante cierto tiempo; para reactivarlas es necesario que el usuario introduzca una contraseña. Algunos teléfonos celulares utilizan un programa el cual consiste de una matriz cuadrada formada por 9 puntos. Para formar su propia contraseña un usuario debe unir todos los puntos mediante 8 líneas. En papel esto podría representarse con ayuda de asteriscos (*) que representen tales puntos y con los caracteres -, |, /, \ y el blanco ' ', para denotar cuando dos puntos se encuentran unidos o no. Una posible contraseña sería la siguiente:

```
* *-*
|/  |
* * *
 /  |
*-*-*
```

Observa que cualquier patrón válido sólo contiene en las líneas 1, 3 y 5 a los caracteres *, - y el blanco, además en cada una de estas líneas las posiciones 1, 3 y 5 siempre contendrán un * y las posiciones 2 y 4 pueden ser – o el blanco. En las líneas 2 y 4 sólo podrá encontrar a los caracteres |, /, \ y el blanco, sin embargo las posiciones 1, 3 y 5 sólo pueden contener el carácter | o el blanco, mientras que las posiciones 2 y 4 consistirán de /, \ o el blanco. Un usuario ha olvidado su contraseña y desea poder recuperarla pero como no es experto en computación ha solicitado tu ayuda. Para facilitarte el trabajo, el usuario te comenta que no todo está perdido, pues recuerda algunas de las líneas que formaban su contraseña. Escribe un programa que genere todas las posibles contraseñas que pudieran formarse a partir de lo que recuerda el usuario. Puedes suponer que siempre habrá solución.

Entrada: Cinco líneas con cinco caracteres cada una representando el patrón incompleto que el usuario puede recordar de la contraseña. Cada una de las líneas puede contener los caracteres *, |, /, \, - y el blanco ' '. Puede suponer que el patrón de la contraseña incompleta siempre será válido.

Salida: Un entero **N** ≥ 1 seguido de las **N** configuraciones encontradas, descritas como en la entrada.

Evaluación: En caso de que todas las configuraciones dadas sean distintas y válidas, 2+8**N**/**T** puntos, donde **T** es la cantidad total de configuraciones válidas.

Ejemplo de entrada	*Ejemplo de salida*

`* *-*`
`    |`
`* *-*`
` /`
`*-*-*`
	`2`
	`*-*-*`
	`|  |`
	`* *-*`
	` /`
	`*-*-*`
	`* *-*`
	`|/  |`
	`* *-*`
	` /`
	`*-*-*`

Problema 3: Colocando variables en memoria (100 puntos)

Código fuente: cvm.c, cvm.cpp, cvm.java

Archivo de entrada: cvm.ent ***Archivo de salida: cvm.sal***

En el lenguaje de programación C, los tipos de datos tienen tanto un tamaño como una alineación en memoria. Cuando un tipo tiene una alineación **A**, las variables de ese tipo sólo podrán ser colocadas en direcciones de memoria que sean múltiplos de **A**. Por ejemplo, si el tipo *int* tiene alineación 4, entonces la dirección de una variable *int* sólo podrá ser alguna de 0, 4, 8, etc.

Suponga que se desean almacenar dos variables, una de tipo *char* (tamaño = 1, alineación = 1) y una de tipo *int* (tamaño = 4, alineación = 4), a partir de la dirección 0 de memoria. Si se coloca primero el *char*, entonces le corresponde la dirección 0 y la siguiente dirección disponible es la dirección 1, sin embargo el *int* no puede ser colocado en ésta y deberá colocarse hasta la dirección 4, siendo la dirección 8 la siguiente disponible. Alternativamente si se coloca primero el *int*, entonces a éste le corresponde la dirección 0 y el *char* puede colocarse en la dirección 4, siendo la dirección 5 la siguiente disponible.

Escribe un programa que, dadas **N** variables a ser almacenadas a partir de la dirección 0, determine en qué orden deben colocarse para que la dirección disponible después de colocarlas sea la menor posible.

Entrada: Un entero **N** seguido de **N** parejas de enteros que denotan el tamaño $\mathbf{T_i}$ y la alineación $\mathbf{A_i}$ de la i-ésima variable a almacenar con $1 \leq i \leq \mathbf{N}$. Puedes suponer que $1 \leq \mathbf{N} \leq 16$, $0 \leq \mathbf{T_i} \leq 16$, $1 \leq \mathbf{A_i} \leq 16$.

Salida: Una secuencia de **N** enteros distintos en el rango de 1 a **N** que denote el orden en el que deben colocarse las **N** variables de la entrada, seguida de un entero **D** que denote la siguiente dirección disponible que resulta de colocarlas en dicho orden.

Evaluación: Si la salida es correcta, 2+8(máx(**B**-**D**,0)+1)/(**B**-**M**+1) puntos, donde **M** es la dirección mínima posible y **B** es la dirección que resulta de colocar las variables en el mismo orden en el que fueron dadas en la entrada.

Ejemplo de entrada	*Ejemplo de salida*
2 1 1 4 4	2 1 5