Los Cuadernos de Trabajo de la Escuela Universitaria de Estadística constituyen una apuesta por la publicación de los trabajos en curso y de los informes técnicos desarrollados desde la Escuela para servir de apoyo tanto a la docencia como a la investigación.

Los Cuadernos de Trabajo se pueden descargar de la página de la Biblioteca de la Escuela www.ucm.es/BUCM/est/ y en la sección de investigación de la página del centro www.ucm.es/centros/webs/eest/

CONTACTO: Biblioteca de la E. U. de Estadística
Universidad Complutense de Madrid
Av. Puerta de Hierro, S/N
28040 Madrid
Tlf. 913944035
buc_est@buc.ucm.es

ISSN: 1989-0567

Laboratorio de Programación.
Manual de Mooshak para el alumno

D. I. de Basilio y Vildósola, M. González Cuñado, C. Pareja Flores

E. U. de Estadística, Universidad Complutense de Madrid

avda. Puerta de Hierro s/n. 18049 - Madrid

e-mail: martagc@emp.ucm.es, dirge20@hotmail.com, cpareja@sip.ucm.es

9 de junio de 2008

Resumen. Los correctores de programas (también llamados jueces) son actualmente herramientas conocidas, usadas ampliamente en el mundo de la enseñanza de la programación. Su modus operandi es sencillo: para un problema planteado, reciben la solución propuesta por un estudiante y comprueban si su funcionamiento es el esperado. En la Universidad Complutense de Madrid, hemos instalado el corrector Mooshak, con el propósito de servir a distintas asignaturas de programación de las facultades de Informática, Matemáticas y Estadística. Este pequeño documento introduce en el desarrollo de programas orientado a las pruebas y explica el manejo de este juez desde el punto de vista del estudiante.

Palabras clave: programación, pruebas de caja negra, corrección automática de programas, concursos de programación.

Celebración del concurso final mundial de programación ICPC 2007

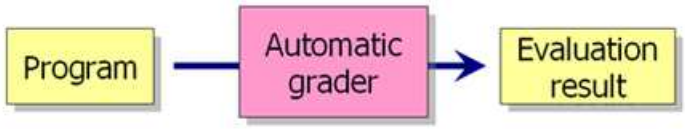
1. Introducción

La actividad central de la enseñanza de la informática es la programación. Se trata de una actividad eminentemente práctica, en que los estudiantes pueden ejercitarse, desarrollando proyectos de menor o mayor envergadura gracias a los compiladores, con bastante autonomía. Un hecho notable es que la programación por sí misma es una actividad con una carga lúdica importante, similar a la matemática recreativa, con la diferencia de que un problema resuelto mediante la programación genera, a su término, el prodigio sorprendente y gratificante de su funcionamiento.

El carácter lúdico ya mencionado de la programación ha dado lugar a la convocatoria de numerosos concursos de programación en el ámbito universitario, a lo largo de todo el mundo. Pero la celebración de concursos se está difundiendo también en el ámbito específico de muchas asignaturas de programación como un ejercicio de laboratorio más. La celebra-

ción de estos concursos y el desarrollo de prácticas controladas, ya sea en el laboratorio o a distancia, desde el hogar de los alumnos, es posible gracias a una clase de herramientas de reciente aparición: los correctores (también llamados jueces) de programas.

Una actividad esencial en el desarrollo de programas es el proceso de prueba, que nos permite detectar errores funcionales, esto es, un funcionamiento incorrecto o distinto del esperado. Los correctores de programas son actualmente herramientas conocidas, usadas ampliamente en el mundo de la enseñanza de la programación. Su modus operandi es sencillo: para un problema planteado, reciben la solución propuesta por un estudiante y comprueban si su funcionamiento es el esperado.



Para ello, compilan y ejecutan el problema sobre una pequeña colección (oculta) de datos de prueba, y comparan si la salida producida por el programa es la esperada o no.

En la Universidad Complutense de Madrid, hemos instalado el corrector automático *Mooshak* [Mooshak, 2008, Leal and Silva, 2003], con el propósito de servir a distintas asignaturas de programación de las facultades de Informática, Matemáticas y Estadística. Este pequeño documento introduce en el desarrollo de programas orientado a las pruebas y explica el manejo de este juez desde el punto del estudiante.

En el siguiente apartado explicamos el estilo de desarrollo de programas que se fomenta con los correctores automáticos de programas. En la sección 4 describimos el laboratorio instalado, y seguidamente indicamos dónde pueden encontrarse otras propuestas de problemas más difíciles, usados concursos de programación. A continuación pasamos a describir el funcionamiento del corrector Mooshak, mediante una visita guiada. Terminamos relacionando algunas de las ventajas más importantes y limitaciones que tiene este enfoque.

2. Programación orientada a las pruebas

La programación orientada o dirigida por los casos de prueba (*test-first*) se ha defendido desde el mundo del desarrollo profesional y en el ámbito académico. En el mundo profesional, este enfoque aboga por que el diseño de un programa empieza por los casos de prueba como ejercicio preliminar con que entender lo que el programa planteado debe realizar y como herramienta primera de especificación de la nueva pieza de código, antes de desarrollar la nueva pieza en sí [Beck, 2001, Shepard, Lamb and Kelly, 2001, XP]. En el ámbito académico, el diseño de los casos de prueba lleva al estudiante a entender mejor la tarea que un programa debe realizar, a prever toda la gama de situaciones que el programa planteado puede encontrar, y que debe atender y, en resumen, a mejorar su propio rendimiento como programador.

Con este laboratorio de programación no perseguimos que los alumnos desarrollen esta metodología, puesto que los problemas que planteamos proporcionan ya un caso de prueba ejemplar, y disponen de otros casos reprueba ocultos. Pero el hábito de leer casos de prueba ejemplares acostumbra también a pensar en la metodología descrita y fomenta una actitud positiva hacia ella. Los estudiantes que preparan un programa para ser revisado por un juez saben que su programa ha de atender toda la gama de situaciones descrita, por lo que están obligados a buscar en dichos programas los recovecos en que puede colarse una situación de error.

Cada problema propuesto, preparado para ser corregido por un juez automático, dispone como decíamos de un juego de datos oculto con que comprobar el correcto funcionamiento del programa presentado por los estudiantes. Estas pruebas se conocen como pruebas de caja negra (black box), porque ignoran el código del programa para observar solamente su funcionamiento. Para cumplir bien con su cometido, dichas pruebas no sólo han de comprobar los casos típicos sino también los atípicos: casos extremos (en los bordes de los vectores, en los límites de precisión), casos singulares (fuera del dominio de las operaciones empleadas, como valores nulos (o cercanos a cero) en los denominadores, negativos en raíces o logaritmos) y situaciones anómalas o degeneradas.

Es imperativo que los estudiantes se habitúen a tener en cuenta todas estas situaciones y **antes de enviar su programa**, presuntamente correcto, lo sometan a un control de calidad teniendo en cuenta los mismos criterios mencionados en el apartado anterior. Dichas pruebas han de ser ahora preparadas por los desarrolladores (los estudiantes), antes de entregárselas al corrector (el cliente) para su revisión. Una decisión típicamente generalizada en estos correctores automáticos es la penalización de las soluciones enviadas que no pasan la batería de pruebas oculta. Es una medida justa y conveniente, porque obliga a los estudiantes a ser ellos quienes se aseguran de la corrección funcional de los programas **antes** de darlos por buenos, asumiendo así la responsabilidad que les corresponde, en vez de delegar dicha responsabilidad en los correctores.

La moraleja es sencilla: que los estudiantes han de tomar buena nota y habituarse a desarrollar pruebas para los casos corrientes y para los casos que bordean los límites de las condiciones expresadas en el enunciado.

Puede alegarse en contra de este enfoque un hecho cierto: que la calidad de un programa no se basa únicamente en la corrección de su funcionamiento observable con respecto al esperado. Existen otros aspectos que este enfoque no valida:

- Por un lado, el uso adecuado de estructuras de control, tamaño adecuado de módulos y número adecuado de parámetros, elección adecuada de estructuras de datos, documentación, uso de las variables necesarias y no más, ausencia de efectos laterales, estructuración adecuada, y un largo etcétera)
- Por otro, la eficiencia, que se puede medir externamente acotando el tiempo y espacio de memoria empleados durante su funcionamiento.

El primer bloque de aspectos de calidad no se contempla en principio en la corrección automática, básicamente porque su revisión depende altamente del lenguaje de programación empleado. La eficiencia en cambio es una responsabilidad del programador, y es posible controlarla en efecto del modo indicado, proporcionando las cotas de tiempo y espacio admitidos para cada solución. Hay que decir sin embargo que el tiempo y espacio empleados también son aspectos dependientes del lenguaje de programación, de forma que Java y Haskell

no pueden someterse con justicia a los mismos patrones que Pascal y C++ por ejemplo. Con los ajustes necesarios para las soluciones presentadas en los distintos lenguajes admitidos, un programa que no supere la prueba por rebasar el tiempo o la memoria tendrá seguramente defectos de diseño más profundos, debidos a la elección de técnicas o estructuras de datos inadecuados para el problema planteado. Así pues, aunque no se indica expresamente en ningún problema, se espera que las soluciones presentadas sean eficientes.

3. Pruebas de caja blanca y caja negra

La corrección de un programa se garantiza mediante técnicas bien distintas: la verificación es una técnica importante y de gran implantación en el mundo académico. Su objetivo es garantizar la corrección de un programa con respecto a su especificación. La garantía viene dada por una demostración matemática, cuyo desarrollo es, en ocasiones, al menos tan difícil como el del programa en sí, y cuya validación (porque una demostración es tan propensa a errores como lo es un programa) es igualmente difícil 👀 **[**[5] [Daly]**]** . La segunda clase de técnicas es la validación de un programa mediante casos de prueba. Un primer enfoque es el de las pruebas de caja blanca (*clear box*): los casos de prueba diseñados han de cubrir todo el código fuente, mostrando que cada fragmento es útil para algún caso de prueba, y que funciona adecuadamente. Este enfoque es útil para una revisión a la vista del código fuente, pero también es difícil llevarlo a cabo de forma automática. Por poner un ejemplo, un programa que resuelve una ecuación de segundo grado podría, quizá, efectuar comprobaciones redundantes (discriminante positivo, por ejemplo) que no son usadas porque ya han sido efectuadas anteriormente. Las pruebas de caja blanca pueden, por ejemplo, poner al descubierto fragmentos de código superfluos como éste. Las pruebas de caja negra (*black box*) comprueban la corrección funcional, esto es, que el programa propuesto funciona adecuadamente, es decir, que el produce los resultados esperados para cada posible juego de datos previsto en el enunciado. Esto exige preparar casos de prueba completos, o sea que abarcan toda la casuística posible, incluyendo las situaciones típicas y atípicas, poniendo en juego valores singulares (por ejemplo, el 0 y 1 como índices de un vector), no definidos (fuera de rango: negativos o demasiado grandes), fuera de los límites de precisión, etc.

Un ejemplo sencillo puede ayudarnos a valorar el trabajo necesario para generar un conjunto de pruebas de caja negra completo: se plantea el pro-blema de resolver ecuaciones de la forma

$$ax^2 + bx + c = 0$$

con coeficientes reales. Un intento de programa, consistente en la mera aplicación de la fórmula conocida para una ecuación de segundo grado, queda desmontado al considerar la casuística posible de un modo exhaustivo:

Ecuaciones de grado menor que 2	0 0 0	Infinitas soluciones
	0 0 1	Sin solución
	0 2 -1	Una solución
Ecuaciones de segundo grado	1 2 0	Dos soluciones reales
	1 2 1	Una solución doble
	1 1 1	Dos soluciones imaginarias

Iniciamos así una solución más completa. Pero las pruebas de caja blanca pueden, por ejemplo, poner al descubierto comprobaciones redundantes (discriminante positivo) que no son usadas porque ya han sido efectuadas anteriormente. Un buen programa ha de emplear todo su código; de lo contrario, tiene código superfluo.

4. Descripción del laboratorio instalado

En el contexto del proyecto PCD08 33 de la UCM, en el departamento de Sistemas Informáticos y Computación de la Universidad Complutense de Madrid, hemos instalado un laboratorio de programación y desarrollado colecciones de problemas de programación para dicho laboratorio. El laboratorio es accesible mediante Internet en la siguiente dirección: `http://problem-g.estad.ucm.es/~mooshak`

El juez corrector instalado es Mooshak [Leal and Silva, 2003]. Por el momento, revisa programas escritos en Pascal, C++ y Java, aunque pronto estará disponible también para

Haskell. Más concretamente, los compiladores subyacentes son los siguientes, en el momento de publicarse este documento:

- Pascal: Free Pascal
- C++: gcc versión 4.1.3, con STL disponible
- Java: Eclipse Java Compiler v_774_R33x, con la API disponible

Las colecciones de problemas preparadas son aptas para su uso en las asignaturas de programación de distintas asignaturas de programación de las facultades de Informática, Matemáticas y Estadística. Son variadas en dificultad y técnicas necesarias para su resolución. A título de ejemplo, describimos las colecciones de ejercicios previstas para una asignatura de introducción a la programación, que es común en contenidos a las facultades mencionadas. Los ejercicios incluidos se agrupan en cuatro colecciones, según las técnicas de programación necesarias para su resolución:

- Introducción: expresiones con tipos de datos básicos y cadenas de caracteres. Instrucciones básicas de lectura, escritura y asignación.
- Instrucciones estructuradas: selección condicional y por casos. Repetición: bucles controlados por un índice y bucles condicionales.
- Subprogramas: procedimientos y funciones. Recursividad.
- Estructuras de datos: vectores y matrices, registros.

Cada colección consta de unos 25 problemas, de dificultad gradual entre cinco categorías, desde los triviales (aptos simplemente para aplicar los mecanismos del lenguaje de programación elegido), hasta los de gran dificultad. Esta dificultad es normalmente de naturaleza algorítmica, pero también a veces se debe a la utilización sutil de los pocos mecanismos del lenguaje permitidos en el momento de su planteamiento. Así por ejemplo, un ejercicio puede resultar de la máxima dificultad en el bloque inicial, pero ser relativamente fácil usando, por ejemplo, bucles.

Lo que podemos garantizar es que todos los ejercicios de cada bloque pueden resolverse con los mecanismos del lenguaje de dicho bloque, y están planteados para resolverlos con sólo dichos mecanismos.

5. Problemas de mayor nivel. Concursos de programación

Habrá estudiantes que se habitúen a aceptar estos pequeños desafíos, y posiblemente se atrevan a probar con otros de mayor dificultad. En las últimas décadas han proliferado multitud de concursos de programación en el mundo, y los problemas planteados en dichos concursos están publicados también en multitud de direcciones de libros, y también en Internet. Es inútil pretender mencionar todos los concursos que funcionan con regularidad en el mundo, por lo que citamos únicamente dos por nuestra relación con ellos:

- El Concurso Universitario de la Comunidad Autónoma de Madrid, [CUPCAM], que se celebra anualmente y cuyos enunciados y soluciones se vienen publicando regularmente en la revista Novática [Novática]

  Los ganadores de este concurso representan a nuestra Comunidad Autónoma en el South-Western European Regional Contest, [SWERC, 2007], en el que participan estudiantes universitarios de España, Francia, Italia, Portugal, Austria occidental, parte de Alemania (Baden - Wurttemberg, Bayern, Berlin, Brandenburg, Mecklenburg-Vorpommern, Sachsen, Sachsen-Anhalt and Thuringen), y suiza.

- El International Collegiate Programming Contest (ICPC), de la Associación for Computing Machinery [ACM ICPC], que es también anual, su ámbito es mundial y tiene la máxima categoría y prestigio entre los concursos de programación.

El ACM ICPC atrae una gran expectación, y los problemas propuestos en ellos son publicados y recopilados en distintas páginas Web del mundo. Entre ellas, hay una que se ha destacado por encima de las demás, por la gran cantidad de enunciados que recoge y por disponer de un juez online, con el que cualquiera puede medir su destreza en programación:

el repositorio de la Universidad de Valladolid [UVA]. Además de éste, el de la universidad de Pekín [Peking University] y la página de las olimpiadas de programación de Estados Unidos [USACO] ofrecen amplias colecciones de problemas de programación de alto nivel y con correctores automáticos de uso libre en Internet.

Ojalá que muy pronto los problemas que planteamos en nuestras colecciones se te queden pequeños y te animes a bucear en estos enlaces, donde encontrarás sin duda retos mucho más interesantes.

6. Discusión

Aunque la preparación y ejecución de este tipo de prácticas exige una experiencia y carga de trabajo considerable por parte del equipo de profesores, presenta una serie de ventajas para los alumnos que la hacen interesante [Hernán, Pareja y Velázquez, 2008]. Citamos las siguientes, entre otras:

- La resolución de estos retos puede realizarse desde cualquier punto, gracias a Internet.
- También gracias a Internet, el momento en que estas prácticas se pueden desarrollar es libre.
- Por tanto, esta actividad fomenta la enseñanza a distancia, si bien es igualmente adecuada para la enseñanza presencial.
- Creando una colección de prácticas que recoja propuestas de distintos niveles de dificultad, podremos atender y fomentar el aprendizaje de alumnos con distintos niveles iniciales, que avanzan a distintas velocidades, o que simplemente se proponen metas con distintos grados de ambición.
- Una conclusión proveniente de las ventajas anteriores es el fomento del autoaprendizaje.

Pero también existen una serie de limitaciones que debemos señalar:

- Ya citábamos al principio la gran carga de trabajo que requiere preparar este tipo de prácticas, por lo que el desarrollo y preparación de problemas se ha de desarrollar en equipo y con apoyo técnico e institucional.
- Este enfoque de desarrollo no cubre adecuadamente todos los tipos de ejercicios y prácticas de programación. Por citar unos pocos tipos, quedan fuera los proyectos que requieran componentes de interfaz de usuario, que sean interactivas, que usen generadores de números aleatorios, etc.

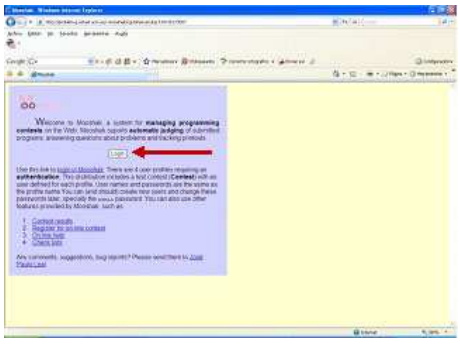
En resumen, el desarrollo de programas fomentado por el uso de correctores automáticos no puede aplicarse a todos los tipos de proyectos de programación imaginables, sino que es adecuado especialmente para problemas de programación de naturaleza algorítmica. Este tipo de problemas sin embargo encierra la principal dificultad de gran parte de los problemas propuestos en muchas asignaturas de programación, concretamente las de los primeros años de las carreras de informática y otras en que la programación es una componente del currículo.

7. Manual Mooshak

7.1. Acceso

Introduce en tu navegador la dirección `http://problem-g.estad.ucm.es/~mooshak`.

Te aparecerá la pantalla de bienvenida al programa. Para acceder a los problemas pulsa sobre la tecla “login”



En la pantalla de autenticación selecciona en la pestaña “contest”la opción que te haya indicado tu profesor.

En el caso de nuestro ejemplo seleccionamos pruebaPrevia.

A continuación introduce tu nombre de usuario y contraseña. **no pulses enter en tu teclado**, haz clic sobre el botón “login”

Existe también la posibilidad de acceder en modo no supervisado, para ello selecciona “cinco problemas” y pulsa sobre ”register”

Introduce el nombre del equipo que vas a crear (*Name*), podrás añadir tantos usuarios como desees. Escribe una dirección de correo valida ya que te enviarán la clave de acceso a dicho correo. Pincha sobre el grupo al que quieras pertenecer (*Group*), y finalmente pulsa sobre ”Submit”

Selecciona el problema que quieres resolver

(En este caso tenemos 5 opciones) Una vez seleccionado el problema aparecerá en la parte inferior de la pantalla

Información sobre entorno de usuario

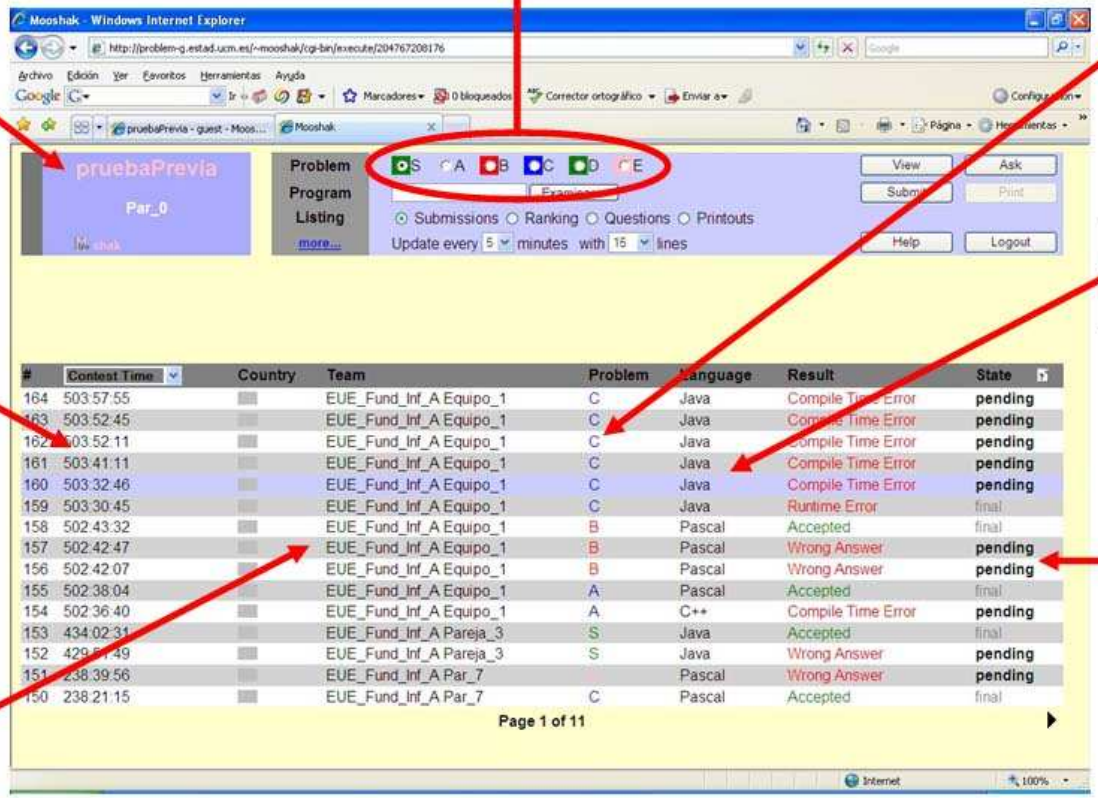
Problema seleccionado para su resolución

Tipo de lenguaje de programación utilizado para solucionar el problema

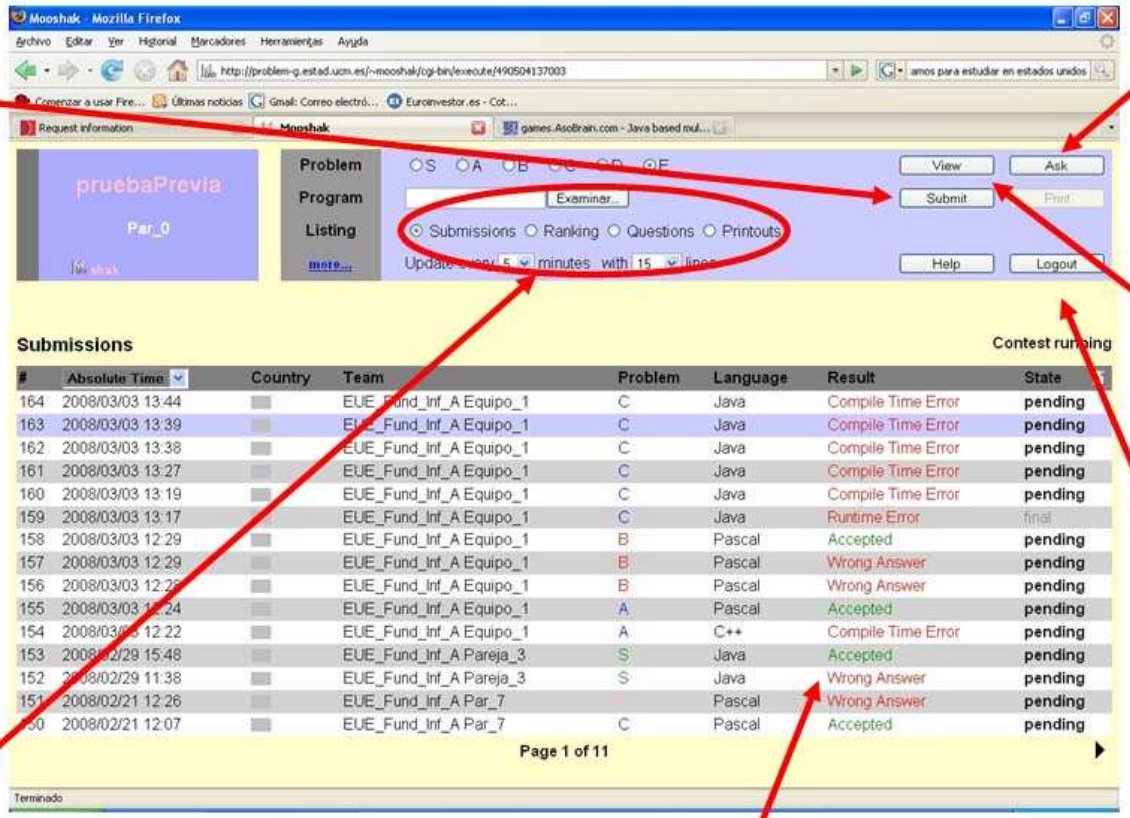
Fecha y hora en la que se envió el la solución al problema

Estado en el que se encuentra el problema, una vez aprobado por el juez (pending) debe ser revisado por el profesor (final)

Usuario que ha enviado la solución del problema



Cuando hayas completado y guardado la solución del problema, pulsa sobre el botón "examinar", selecciona el problema desde la ubicación en la que lo hayas guardado y a continuación pulsa sobre el botón "Submit" para enviarlo



Para cualquier consulta que necesites realizar al profesor pulsa el botón Ask"

Desde cualquier ventana en la que te encuentres, si deseas volver a ver el enunciado del problema que has seleccionado, pulsa sobre el botón "View"

- Submisions: Problemas enviados y su estado
- Ranking: Puesto que ocupa cada usuario
- Questions: Preguntas realizadas por los usuarios con las respuestas de los profesores.
- Printouts: Información sobre usuarios y horas a las que se ha imprimido

En esta columna encontrarás la información sobre el resultado del problema que has enviado. Si es correcto aparecerá en verde "Accepted", de lo contrario los posibles errores más comunes son:

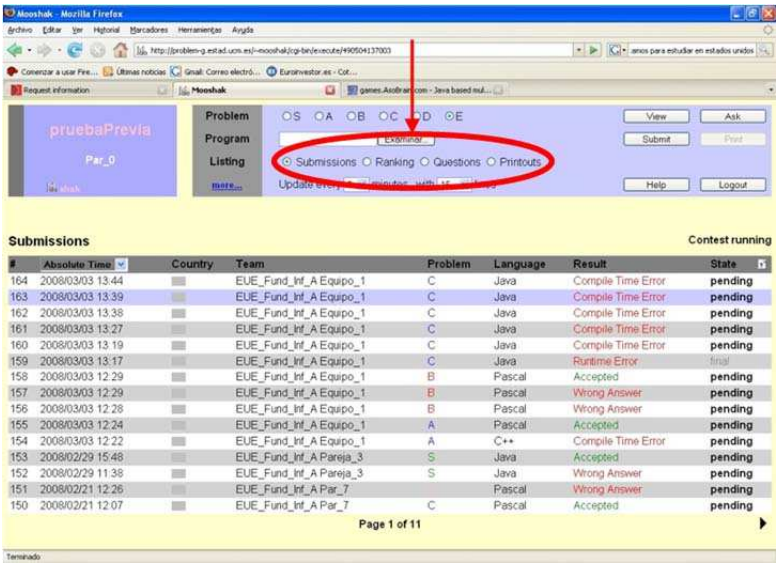
- Compile Time Error: El programa no compila
- Wrong Answer: El programa ha dado una respuesta incorrecta
- Runtime Error: El programa no funciona correctamente, "se cuelga"
- Presentation error: El programa parece correcto pero la salida no está en el formato adecuado.

Para más información sobre diferentes errores pulsa sobre el botón "Help"

Cuando acabes de trabajar desconecta tu sesión pulsando sobre "logout"

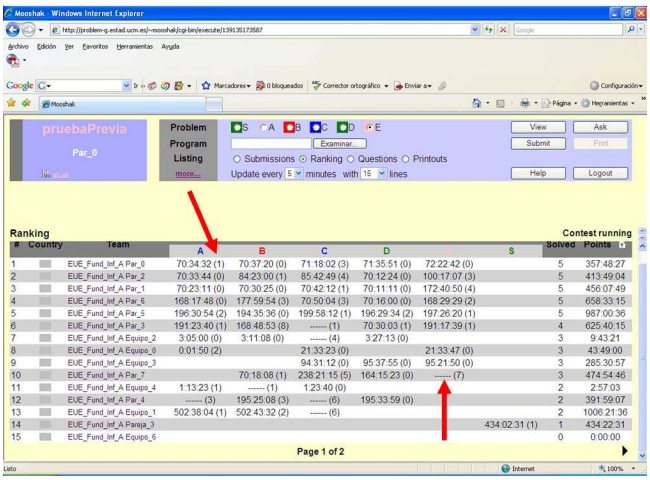
7.2. Contenido

Veamos ahora más detalladamente las ventanas principales.



- **Submisions:** Como ya indicamos anteriormente, es la pantalla inicial donde encontraras la información general referente a usuarios y envío de problemas.

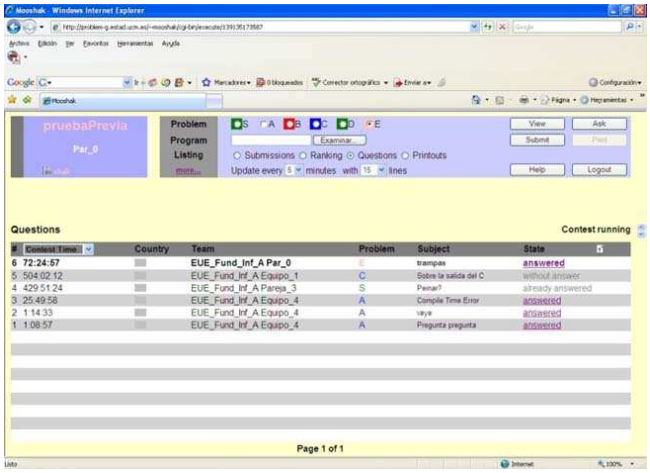
- **Ranking:** El puesto que ocupa cada usuario en el Ranking depende del tiempo transcurrido desde que empezó el concurso hasta el envío del problema, de la cantidad de veces que se ha enviado hasta que el juez y los profesores aceptan la solución y de la cantidad de problemas solucionados que se han enviado.



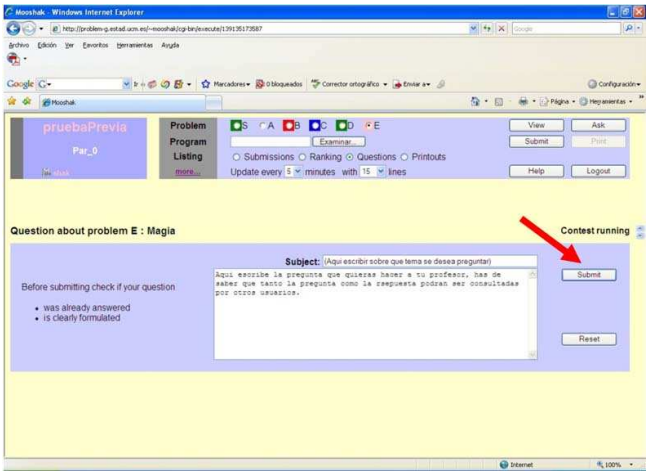
Junto al tiempo, entre paréntesis, aparece la cantidad de veces que se ha enviado el problema, si no aparece el tiempo es que aún no se ha dado una respuesta correcta. Los espacios en blanco son problemas no enviados todavía.

- **Questions:** En este apartado encontrarás preguntas de otros usuarios sobre los problemas.

Como te indica el programa es un apartado público, puedes consultar las existentes o enviar tu propia pregunta.



Selecciona el problema sobre el que quieres hacer la pregunta, pon un título que lo resuma, escribe tu pregunta y pulsa sobre el botón “submit”

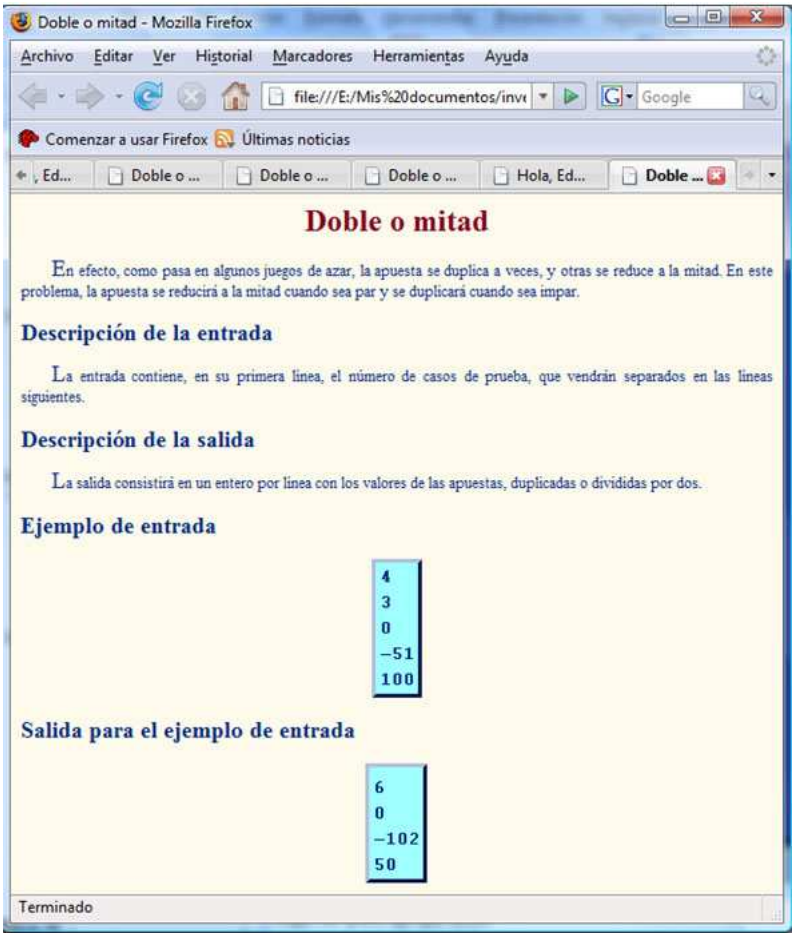


Asegúrate de que tu pregunta **no** está ya respondida e intenta ser lo más claro y específico posible al redactarla.

Posibles Errores

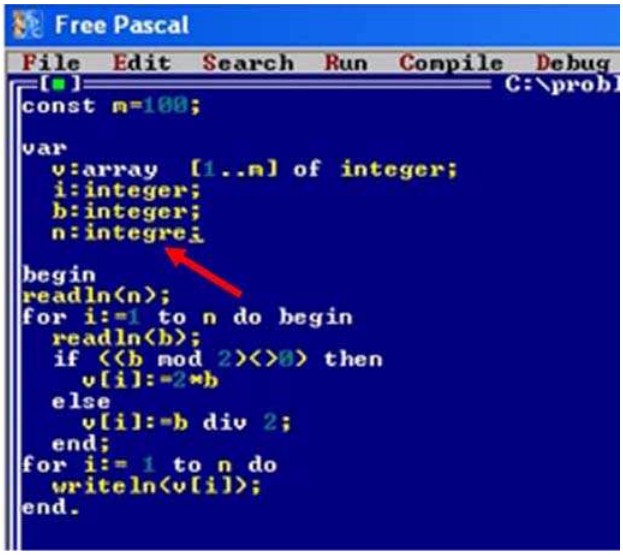
Veamos ahora algunos ejemplos de errores que pueden cometerse a la hora de solucionar un problema.

Tomamos como ejemplo el siguiente:

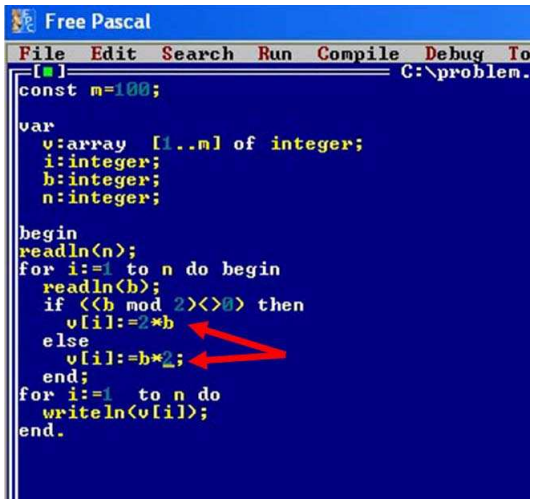


- **Compile Time Error:** El programa no compila

En este caso deberíamos haber escrito `integer` en lugar de `integre`



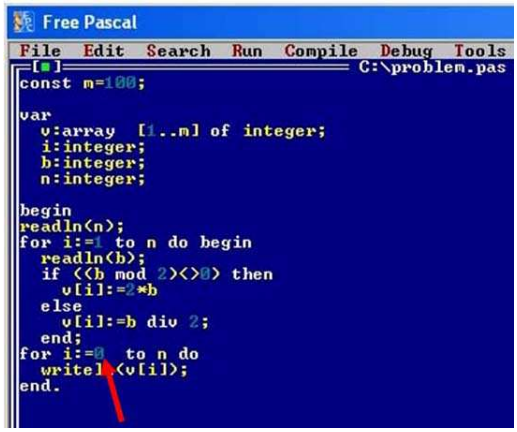
- **Wrong Answer:** El programa ha dado una respuesta incorrecta



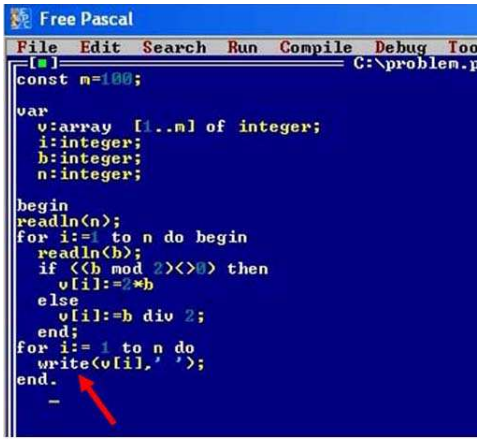
En este caso damos proponemos la misma fórmula para ambos supuestos, pares o impares, por lo tanto la solución que le estamos dando no se ajusta al enunciado del problema, que nos pedía un comportamiento diferente para pares e impares.

- **Runtime Error:** El programa no funciona correctamente, "se cuelga"

En este caso la variable `v[i]` hace una llamada a un valor que no existe, (hemos definido el vector `v` de `1` a `n`, por lo tanto el programa no se ejecuta correctamente.



- **Presentation error:** El programa parece correcto pero la salida no está en el formato adecuado.



En el caso de nuestro ejemplo, deberíamos haber escrito `writeln` en lugar de `write` y sobran las comillas

Agradecimientos

Este documento se ha desarrollado en el contexto del proyecto PCD08 33, financiado con fondos del Vicerrectorado de Innovación y Espacio Europeo de Educación Superior, de la Universidad Complutense de Madrid, destinados a los Proyectos de Innovación y Mejora de la Calidad Docente, en el ejercicio de 2008.

Referencias

[Beck, 2001] K. Beck, *Aim, fire (test-first coding)*. IEEE Software, num.18 vol. 5, sept.-oct. 2001, pp. 87-89.

[CUPCAM] Concurso Universitario de Programación de la Comunicad Autónoma de Madrid, `http://www.cupcam.es/`.

[Hernán, Pareja y Velázquez, 2008] I. Hernán, C. Pareja, J. Á. Velázquez, *Testing-Based Automatic Grading: A Proposal from Bloom's Taxomony*, The 8^{th} IEEE International Conference on Advanced Learning Technologies, ICALT 2008.

[XP] *Extreme Programming. A gentle introduction*, `http://www.extremeprogramming.org`. Last modified feb. 2006.

[ACM ICPC] ACM *International Collegiate Programming Contest* (ICPC), `https://cm2prod.baylor.edu/`.

[Leal and Silva, 2003] José Paulo Leal and Fernando Silva, Mooshak: a Web-based multi-site programming contest system. *Software Practice & Experience*, 33(6), 567-581, 2003.

[Mooshak, 2008] Mooshak. `http://mooshak.dcc.fc.up.pt/`, último acceso en abril de 2008.

[Novática] Novática, `http://www.ati.es/novatica/`, sección "Programar es crear".

[UVA] Online Judge System and Problem Set Archive, Universidad de Valladolid. `http://acm.uva.es/problemset/`

[Peking University] The Peking University Judge Online: http://acm.pku.edu.cn/JudgeOnline/

[Skiena y Revilla, 2006] Steven Skiena y Miguel Revilla, Concursos Internacionales de Informática y Programación. “Manual de entrenamiento por Internet” http://www.programming-challenges.com/pg.php?page=index.

[Shepard, Lamb and Kelly, 2001] T. Shepard, M. Lamb and D. Kelly, ”More testing should be taught”. *Communications of the ACM* 44(6): 103-108, June 2001.

[SWERC, 2007] *South-Western European Regional Contest*, SWERC, http://icpc.baylor.edu/icpc/Regionals/SWERC07/, edición de 2007.

[USACO] USA Computing Olympiads, ttp://www.uwp.edu/sws/usaco/, Training Program Gateway, http://www.uwp.edu/sws/usaco/

Apéndice. Cinco problemas básicos

Una vez visto el funcionamiento general del juez Mooshak, te proponemos algunos problemas sencillos para ponerlo en práctica. Los cinco problemas están elegidos de un repertorio para un curso introductorio de programación. El primero de ellos (Matriz por columnas) puede resolverse usando únicamente instrucciones básicas (lectura, escritura y asignación, con tipos de datos básicos), aunque algunos estudiantes podrán quizás encontrarlo más sencillo si emplean también bucles; no hace falta usar matrices, a pesar de su título. El segundo (Factorial) es típico; necesita bucles o subprogramas recursivos. El tercero (Conjetura de Goldbach) está planteado para resolverse con subprogramas, aunque se puede afrontar (si bien más engorrosamente) usando únicamente instrucciones estructuradas. El problema del punto de silla requiere matrices. El último (ordenación) necesita el uso de vectores.

Problema A

Matriz por columnas

`MatrizColumna.{pas,cpp,java}`

Si rellenamos na matriz de 4 x 7 con los números naturales, de columna en columna,

1	5	9	13	17	21	25
2	6	10	14	18	22	26
3	7	11	15	19	23	27
4	8	12	16	20	24	28

podemos saber la posición de un elemento cualquiera. De eso trata el programa que planteamos ahora: partiendo de las dimensiones (número de filas y columnas) de una matriz y un elemento, nuestro programa deberá averiguar su posición: número de la fila y columna en que se encuentra.

Descripción de la entrada

La entrada consiste en tres números enteros positivos, en una misma línea, separados por espacios simples en blanco.

Descripción de la salida

La salida es un par de enteros, separados por un espacio simple, representativos de la fila y columna en que se encuentra el elemento dado.

Ejemplo de entrada

```
4 7 15
```

Salida para el ejemplo de entrada

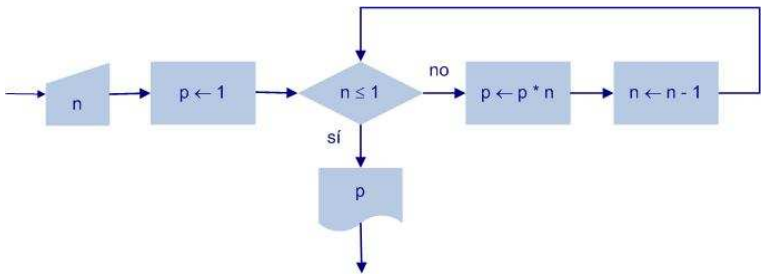
```
3 4
```

Problema B

Factorial

`Factorial.{pas,cpp,java}`

Aunque el enunciado te da una pista, no te confíes: lo que te pedimos es un programa que desempeñe lo descrito en el siguiente diagrama de flujo:



Descripción de la entrada

La entrada contiene, en su primera línea, el número de casos de prueba, que vendrán separados en las líneas siguientes. Todos los datos serán enteros positivos.

Descripción de la salida

La salida consistirá en un entero por línea con los resultados de los casos introducidos.

Ejemplo de entrada

```
4 3 0 10 5
```

Salida para el ejemplo de entrada

```
6 1 3628800 120
```

Problema C

Conjetura de Goldbach

`Goldbach.{pas,cpp,java}`

En una carta a Leonard Euler en 1792, Christian Goldbach afirmó (sin demostrarlo), que todo número par es la suma de dos números primos. Te pedimos un programa que acepte un número par como entrada y escriba un par de primos cuya suma iguale al par dado. En caso de haber varias descomposiciones como la descrita, el programa ofrecerá aquélla en que el primer primo dado sea el menor posible.

Descripción de la entrada

La entrada contiene un único número natural par mayor que dos.

Descripción de la salida

La salida ha de contener dos números naturales en una única línea, separados por un espacio simple.

Ejemplo de entrada

```
30
```

Salida para el ejemplo de entrada

```
7 23
```

Nota: Esta conjetura aparece recogida en las Meditaciones algebraicas de Edward Waring (1734-1793) junto con otros muchos resultados interesantes. En la novela El tío Petros y la conjetuda de Goldbach, de A. Doxiadis (Ediciones B, 2000), se relata la historia de un matemático que únicamente vivió para intentar demostrar la conjetura de Goldbach. Esta conjetura también aparece en la película española La habitación de Fermat (2007), dirigida por Luis Piedrahita y Rodrigo Sopeña.

Problema D

Punto de silla

PuntoSilla.{pas,cpp,java}

Un punto de silla de una matriz es un elemento que es el menor estrictamente de su fila y el mayor estrictamente de su columna. No todas las metrices tienen un elemento con esta propiedad, pero cuando una matriz lo tiene, es único. Se pide un programa que busca un punto de silla en una matriz, e indica valor o la palabra NO, en caso de que no exista.

Descripción de la entrada

La entrada contiene una matriz de 5 x 3 enteros, organizados por filas del modo usual, y separados por blancos simples.

Descripción de la salida

La salida es una única línea. En caso de que la matriz de la entrada contenga el punto de silla, la salida ha de contener tres enteros separados por un espacio simple: su posición (fila y columna) y el valor; en caso de que no exista punto de silla en dicha matriz, basta con la palabra NO.

Ejemplo de entrada

```
1 2 3 -4 -5
-6 7 -8 4 0
20 15 10 5 13
5 5 -5 2 -500
```

Salida para el ejemplo de entrada

```
1 2 3 -4 -5
-6 7 -8 4 0
20 15 10 5 13
5 5 -5 2 -500
3 4 5
```

Problema E

Ordenación simple

`Ordenar.{pas,cpp,java}`

Simplemente, se trata de ordenar una lista de enteros ascendentemente.

Descripción de la entrada

La entrada contiene 10 enteros, en una única línea, y separados por blancos.

Descripción de la salida

La salida ha de contener los diez enteros de la entrada, pero de menor a mayor, y separados por un espacio simple.

Ejemplo de entrada

```
17 4 7 1 5 3 0 -1 1000 0
```

Salida para el ejemplo de entrada

```
-1 0 0 1 3 4 5 7 17 1000
```