ENTREGA Global (3 puntos)

Antes de empezar...

- Descargar el archivo **EGlobal.zip** e importarlo a Eclipse.
- Renombrar el proyecto (tecla F2). Por ejemplo, para la estudiante Ana Pérez el nombre del proyecto sería: **EGlobal_APerez**.

Además, ten en cuenta que:

- Se valorará la eficiencia de las soluciones, es decir, además de que sean correctas, deben ser eficientes.
- Aparte de los casos de prueba que se os entregan, se espera que incorpores casos de prueba adicionales y significativos.
- Para entregarlo, debes exportar el proyecto y subirlo a eGela.

Fecha límite de entrega: 16/01/2024

—

Ayuda para importar:

File -> Import... -> General -> Existing Projects into Workspace -> Select archive file (el .zip descargado) -> Finish

Ayuda para exportar:

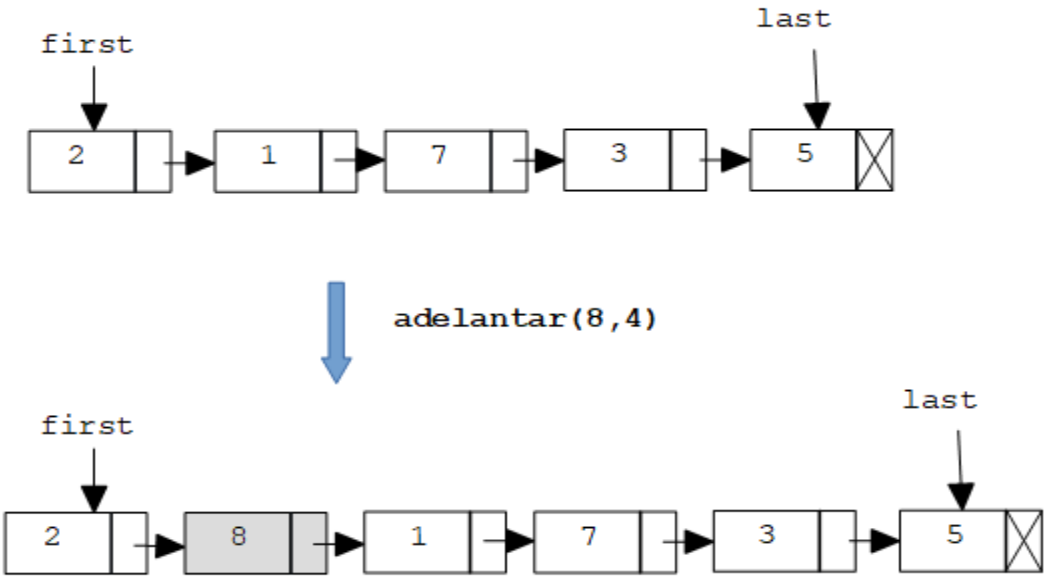
Pinchar en el proyecto. File -> Export... -> General -> Archive File -> (seleccionar las carpetas/archivos a exportar) -> To archive file (escribir una ruta y nombre para el nuevo archivo .zip) -> Finish

Ejercicio 1 (listas)

Una cola de enteros se puede representar mediante una lista simplemente encadenada con apuntador al primer y último elemento.

Se pide implementar el método **adelantar(int elem int n)** de la clase **Cola**. Este método inserta el elemento *elem* en la cola pero *n* posiciones antes de lo que le correspondería. Si la cola tiene menos de *n* elementos no hace nada, y si *n* es 0 inserta *elem* al final de la cola. Además devuelve un booleano que indica si se ha podido insertar el elemento en la cola o no.

Ejemplo:



Ejercicio 2 (listas)

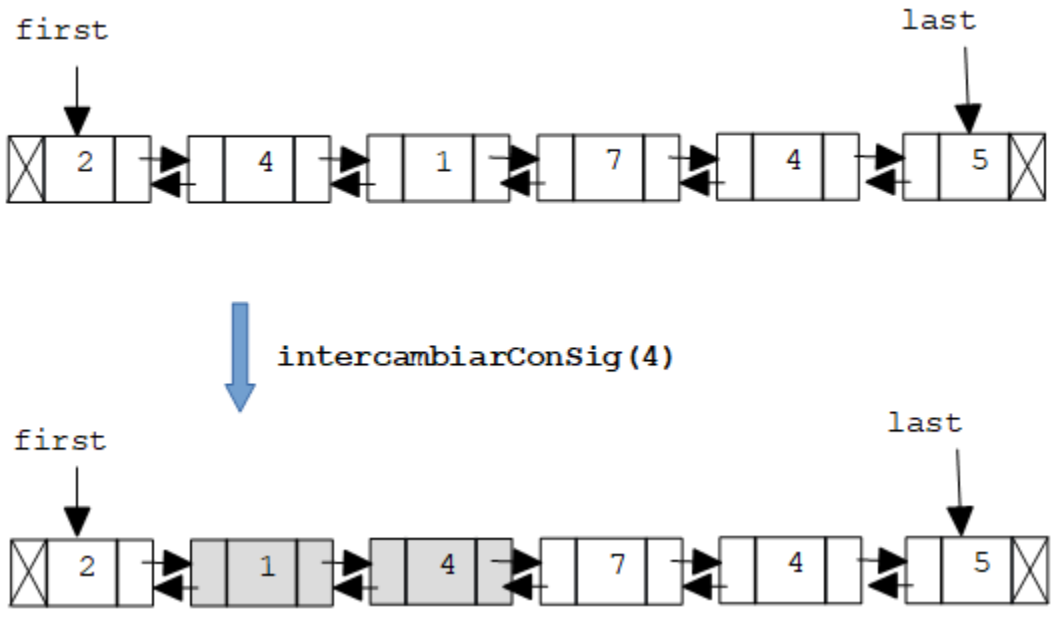
La clase **DoublyLinkedList<T>** sirve para representar listas doblemente encadenadas, con apuntador al primer y último elemento.

Se pide implementar el método **intercambiarConSig (T elem)**. Este método intercambia el nodo que contiene la primera aparición de *elem* con el siguiente nodo de la lista.

Notas:

- Supondremos como precondición que *elem* está en la lista y que no está en la última posición.
- El intercambio se debe hacer moviendo los nodos, es decir, no vale, por ejemplo, intercambiar los valores de los nodos, ni crear copias de los nodos.

Ejemplo:



Ejercicio 3 (pilas y colas)

Queremos gestionar el peaje de una autopista. Cada coche que llega al peaje trae un ticket que habrá cogido al entrar a la autopista. Disponemos de la clase *Ticket*, con la siguiente estructura:

```
public class Ticket {
  String matricula; //matrícula del coche
  double tEntrada; //en qué instante (en segundos) cogió el ticket
  double tPeaje; // en qué instante (en segundos) ha llegado al peaje
  double distanciaRecorrida; // distancia recorrida desde que cogió el
                             //ticket hasta que ha llegado al peaje (en metros).
}
```

En el peaje hay varias cabinas de pago, y en cada una de ellas se registran los tickets de los coches que llegan a dicha cabina:

```
public class Peaje { //suponemos getters y setters
  int numCabinas; // nº de cabinas del peaje
  ArrayList<Queue<Ticket>> registro; // Estructura que registra los
       //tickets de todos los coches que llegan al peaje. Cada elemento (cola)
      //del ArrayList hace referencia a una cabina y guarda los tickets de
      //los coches que llegaron a dicha cabina, ordenados en base a tPeaje.
}
```

Se pide implementar los siguientes métodos de la clase Peaje:

(a) `public Peaje(int numCabinas){…} //TO DO`

Método constructor que crea un peaje que tiene *numCabinas* cabinas preparadas para recibir tickets (aunque aún no han recibido ninguno)

(b) `public ArrayList<Ticket> infractores( ) {…} //TO DO`

El método *infractores* devuelve un ArrayList que contiene aquellos tickets del registro en los que se ha superado la velocidad máxima permitida. Se habrá superado la velocidad máxima si teniendo en cuenta la distancia recorrida en el intervalo de tiempo entre *tEntrada* y *tPeaje* vemos que la velocidad supera los 120km/h. Recuerda que la distancia está dada en metros y los tiempos en segundos.

El método considerará los tickets de todas las cabinas. El ArrayList resultante **deberá estar ordenado en base al atributo *tPeaje* de los tickets, independientemente de la cabina de la que provengan.**

NOTA: no se permite obtener los tickets en cualquier orden y luego ordenarlos, se deben ir obteniendo en el orden correcto.

Para el ejemplo de prueba:

Los infractores son:

Matrícula: 2222-ldb: 168.75km/h

Matrícula: 1234-fkb: 187.5km/h

Matrícula: 9876-lvk: 146.34146341463415km/h

Matrícula: 5678-dbs: 130.38229376257547km/h

Matrícula: 4321-szp: 128.31683168316832km/h

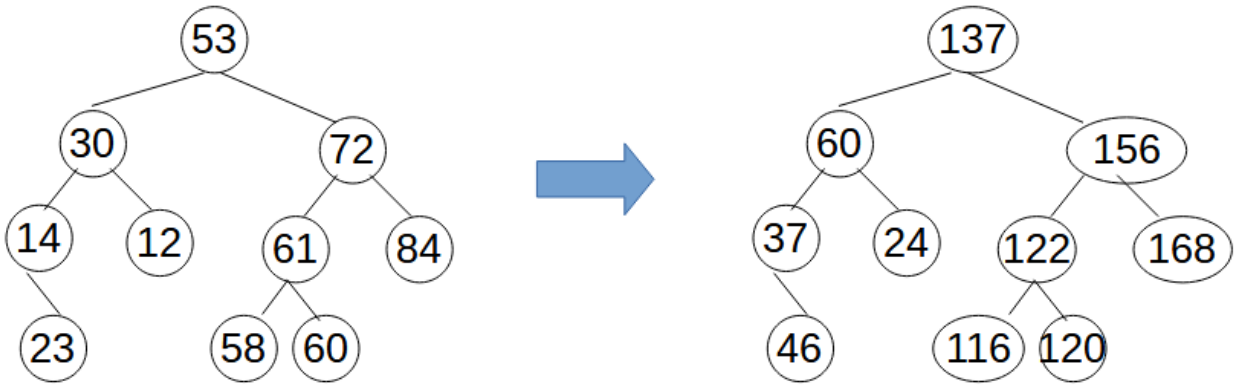
Matrícula: 0000-nyd: 134.8993288590604km/h

Matrícula: 1111-twc: 120.6km/h

Matrícula: 9999-vdf: 148.6238532110092km/h

Ejercicio 4 (Árboles)

Para un árbol binario de enteros, implementar el método **modificarNodo**. Este método modifica el valor del nodo, sumándole el mayor elemento del subárbol que comienza el dicho nodo (para determinar el mayor elemento se tendrán en cuenta los valores originales de los nodos, no los modificados).



En el ejemplo anterior, el nodo que originalmente tenía el valor 72 toma el valor 156, porque se le ha sumado el mayor nodo del subárbol que comienza en el 72: 72+84 = 156. El nodo que originalmente tenía el valor 30 ahora toma el valor 60 porque el propio nodo es el que tenía mayor valor en ese subárbol.

Para los ejemplos de prueba: (ver dibujos al final del enunciado):

```
-------------
 ÁRBOL 0
-------------
Árbol original:
*
Árbol modificado:
*

-------------
 ÁRBOL 1
-------------
Árbol original:
[ 8 ]
Árbol modificado:
[ 16 ]

-------------
 ÁRBOL 2
-------------
Árbol original:
[ 53 [ 30 [ 14 * [ 23 ] ] [ 12 ] ] [ 72 [ 61 [ 58 ] [ 60 ] ] [ 84 ] ] ]
Árbol modificado:
[ 137 [ 60 [ 37 * [ 46 ] ] [ 24 ] ] [ 156 [ 122 [ 116 ] [ 120 ] ] [ 168 ] ] ]
```

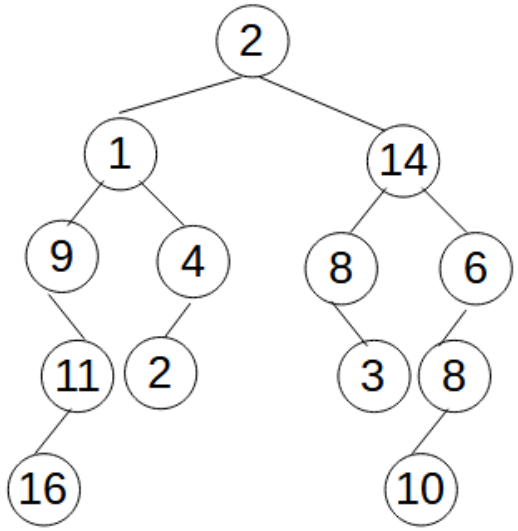
```
-------------
 ÁRBOL 3
-------------
Árbol original:
[ 80 [ 23 * [ 14 [ 13 ] [ 8 ] ] ] [ 18 [ 17 ] [ 22 ] ] ]
Árbol modificado:
[ 160 [ 46 * [ 28 [ 26 ] [ 16 ] ] ] [ 40 [ 34 ] [ 44 ] ] ]

-------------
 ÁRBOL 4
-------------
Árbol original:
[ 55 [ 6 [ 4 [ 2 ] * ] * ] [ 47 [ 18 [ 17 ] [ 19 ] ] [ 21 ] ] ]
Árbol modificado:
[ 110 [ 12 [ 8 [ 4 ] * ] * ] [ 94 [ 37 [ 34 ] [ 38 ] ] [ 42 ] ] ]

-------------
 ÁRBOL 5
-------------
Árbol original:
[ 16 [ 6 [ 4 ] * ] [ 22 [ 18 [ 17 ] [ 19 ] ] [ 21 ] ] ]
Árbol modificado:
[ 38 [ 12 [ 8 ] * ] [ 44 [ 37 [ 34 ] [ 38 ] ] [ 42 ] ] ]

-------------

 ÁRBOL 6

-------------

Árbol original:

[ 16 [ 8 [ 4 [ 2 ] [ 6 ] ] [ 12 [ 9 ] [ 14 [ 13 ] [ 15 ] ] ] ] * ]

Árbol modificado:

[ 32 [ 23 [ 10 [ 4 ] [ 12 ] ] [ 27 [ 18 ] [ 29 [ 26 ] [ 30 ] ] ] ] * ]
```

Ejercicio 5 (Árboles)

Para un árbol binario de enteros, implementar el método **cumple().** Este método devuelve una lista con aquellos nodos del árbol para los que se cumple la siguiente condición:

- El nº de ascendientes del nodo que tienen valor par es igual al nº de descendientes de dicho nodo que tienen valor par.

Por ejemplo, para el siguiente árbol, el método devolvería la lista **<11, 9, 4, 6>** (en cualquier orden)



Para los ejemplos de prueba:(ver dibujos al final del enunciado)

```
-------------
 ÁRBOL 0
-------------
[8]

-------------
 ÁRBOL 1
-------------
[5, 4]

-------------
 ÁRBOL 2
-------------
[9, 1, 3, 5, 7]

-------------
 ÁRBOL 3
-------------
[]

-------------
```

```
 ÁRBOL 4
-------------
[7]

-------------
 ÁRBOL 5
-------------
[11, 9, 4, 6]
```

Ejercicio 6 (Tablas hash)

Llega la hora de realizar los exámenes globales y desde Decanato han pedido a los coordinadores de cada asignatura que realicen una solicitud indicando el nombre de su asignatura y el número de aula donde le gustaría realizar el examen (supondremos un único aula para cada asignatura).

Un vez recibidas todas las solicitudes, desde Decanato realizan las reservas de aula correspondientes, sabiendo que:

- Las solicitudes se procesan en orden de llegada
- Hay exámenes de lunes a viernes
- Cada día hay tres turnos de examen (a las 9:00, a las 12:00 y a las 15:00)
- El periodo de exámenes durará tantas semanas como sea necesario
- A cada asignatura se le asignará el aula deseada en el primer hueco libre para dicho aula.

Debes implementar el método **crearTablaReservas**, que devuelve un HashMap donde para cada aula se indican las reservas hechas (asignatura, nº de semana, día de la semana, hora).

Por ejemplo, para la siguiente lista de solicitudes:

```
<("PB","2.6"); ("PDSD","1.2"); ("FTC","2.6");  ("MD","2.6");
("AM","3.16"); ("EC","2.6");   ("ALG","2.6"); ("CAL","1.2");
("PMOO","2.6"); ("MP","2.6"); ("EAE","2.6");  ("MEI","2.6");
("AC","2.6"); ("LCSI","2.6"); ("IS1","2.6"); ("BD","1.2");
("IO","2.6"); ("ISO","2.6");  ("IRC","2.6");  ("SAR","3.16");
("CC","2.6"); ("MAC","1.2")>
```

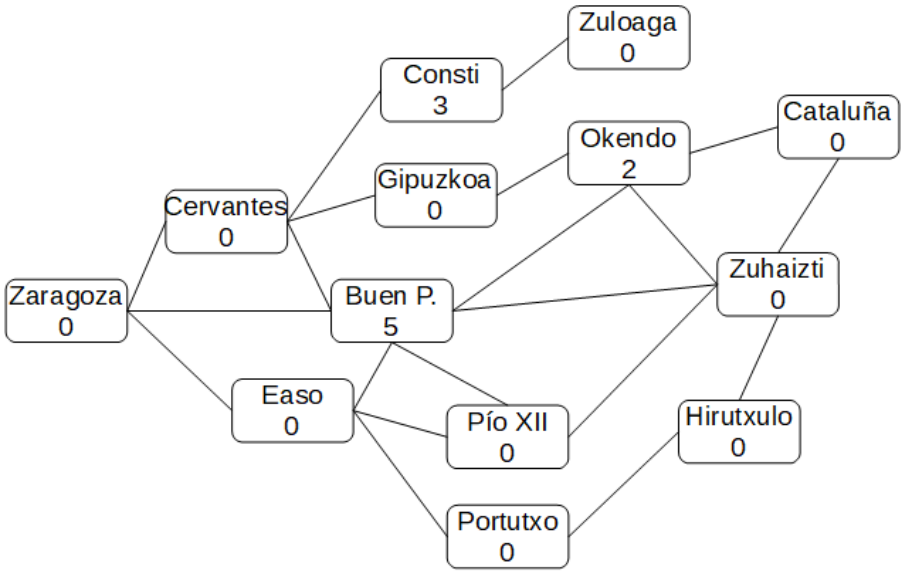
El HashMap creado será:

(clave)	**(valor)**
1.2	[<PDSD,1,L,9:00>, <CAL,1,L,12:00>, <BD,1,L,15:00>, <MAC,1,M,9:00>]
3.16	[<AM,1,L,9:00>, <SAR,1,L,12:00>]
2.6	[<PB,1,L,9:00>, <FTC,1,L,12:00>, <MD,1,L,15:00>, <EC,1,M,9:00>, <ALG,1,M,12:00>, <PMOO,1,M,15:00>, <MP,1,X,9:00>, <EAE,1,X,12:00>, <MEI,1,X,15:00>, <AC,1,J,9:00>, <LCSI,1,J,12:00>, <IS1,1,J,15:00>, <IO,1,V,9:00>, <ISO,1,V,12:00>, <IRC,1,V,15:00>, <CC,2,L,9:00>]

Donde L hace referencia a lunes, M a martes, X a miércoles, J a jueves, y V a viernes. Por ejemplo `<PMOO,1,M,15:00>` significa que el examen de PMOO se realizará en la semana 1, concretamente el martes a las 15:00.

Ejercicio 7 (grafos)

Se tiene un grafo no dirigido, implementado mediante listas de adyacencias, en el que los nodos representan plazas de Donostia y las aristas caminos entre dichas plazas.



En cada nodo se guarda el nombre de la plaza y un entero que indica el nº de cámaras de seguridad que hay en dicha plaza.

Un ladrón quiere ir desde una plaza origen hasta una plaza destino sin ser captado por las cámaras (suponemos que las cámaras abarcan únicamente la plaza entera en la que están situadas y que no hay cámaras en las plazas de origen y destino).

Se pide implementar el método **huida** de la clase GrafoPlazas. Este método recibe una plaza de origen y una plaza de destino y devuelve la longitud del camino más corto (entendido como el camino que pasa por menos plazas) que le permitiría ir desde origen a destino sin ser captado por las cámaras. En caso de que no sea posible, devolverá -1.

En el ejemplo de la figura, si la plaza de origen fuera “Zaragoza” y la plaza de destino “Zuhaizti”, el método debería devolver 3, que es la longitud del camino Zaragoza → Easo → Pío XII → Zuhaizti. Sin embargo, si la plaza de origen fuera “Zuloaga” y la plaza de destino “Gipuzkoa” debería devolver -1, porque no es posible ir de una a otra sin ser captado por las cámaras.

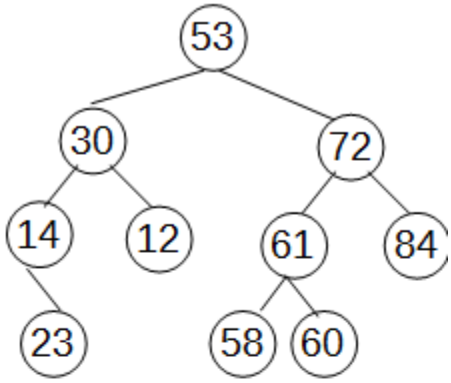
Dibujos de los árboles para el ejercicio 4 (modificarNodo en árbol binario)

arbol0.txt

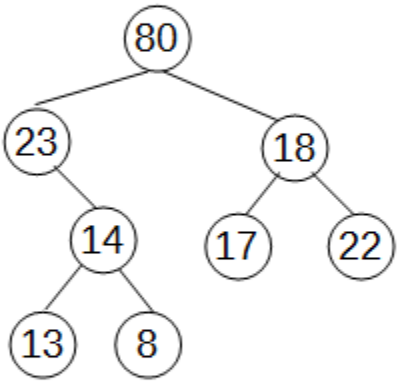
(vacío)

arbol1.txt

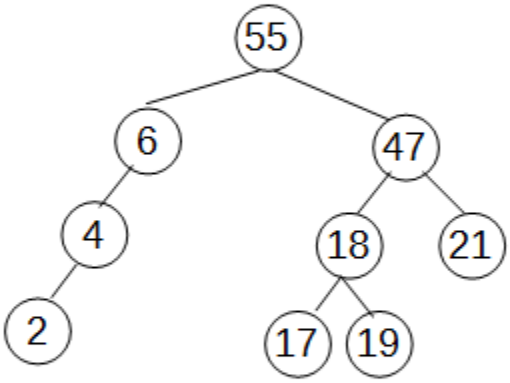
arbol2.txt



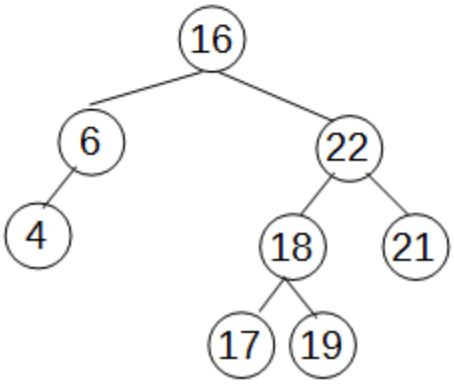
arbol3.txt



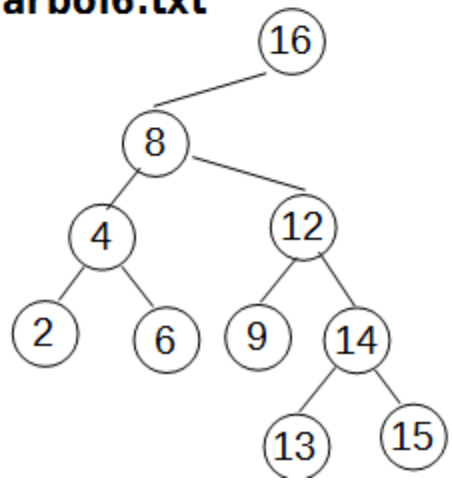
arbol4.txt



arbol5.txt



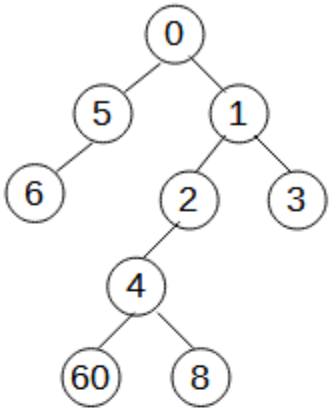
arbol6.txt



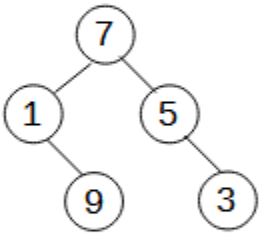
Dibujos de los árboles para el ejercicio 5 (método cumple())

arbol0.txt

arbol1.txt



arbol2.txt



arbol3.txt

(vacío)

arbol4.txt

arbol5.txt