Rodrigo Pacheco Martinez-Atienza

DG. Ing. Tec. Teleocomunicaciones e Ing. Aeroespacial en Aeronav
Ingeniería en Sistemas de la Información
Introducción a pruebas SW (testing)

Para cada programa se ha creado un script de shell que compila y ejecuta los test de forma automática.

FindLast.java

1. Explica el fallo (qué está mal en el código). Describe de manera precisa el fallo y propón una modificación al código que lo arregle.
En la función FindLast los argumentos del for “for (int i=x.length-1; **i >0** 0; i--)” comienzan en la última posición pero no recorren el array hasta la primera posición (0) ya que dicrurren hasta i>0.
Propeusta de corrección: for (int i=x.length-1; **i >=0**; i--). Añadidendo el >= ahora
2. Proporciona, si ello es posible, un caso de prueba que no ejecute el código que tiene el fallo. Si no es posible, explica por qué.
No hay ningún caso en el que no se ejecute el código ya que no es posible lanzar el programa sin argumentos y, desde el momento en el que se lanza el programa, se ejecuta la función FinLast y, consecuentemente, la inicialización del bucle for. En el ejemplo del siguiente apartado se proporciona un caso en el que el código que contiene el fallo sí se ejecuta (inicialización del for) pero no llega al caso de la i errónea.
La única excepción sería ejecutar el código sin argumentos del array y en ese caso, como salta el mansaje de adbertencia de uso del programa no llega a ejecutarse la función FindLast.
3. Si es posible, proporciona un caso de prueba que ejecuta el fallo que hay en el código, pero que no provoque un error en el estado. Si no se puede, explica por qué.
Ejemplo) java FinLast 1 2 3 4
Enter an integer you want to find: 3
El programa primero consultará la posición 3 del array (cuyo contenido es 4), como no es el calor que se busca saltará a la siguiente iteración, consultará la posición del array (contenido 3) y, dado que es el valor buscado, devolverá esta posición. El valor de la variable i será siempre correcta porque no llega a la última posición. El código se ejecuta pero esto no acarrea un estado incosistenete del programa.
4. Si es posible, proporciona un caso de prueba que provoque un error en el estado, pero que no acabe provocando una disfunción en el comportamiento del programa. No olvides que el contador de programa forma parte, junto a las variables, del estado del programa. Si no es posible, explica por qué.
Ejemplo) java FinLast 1 2 3 4
Enter an integer you want to find: 333333333333333333
El programa pasa por un estado incosistenete porque no se comprueba la última posición, pero dado que el número a buscar no está en ninguna posición del array, el resultado final es el mismo independientemente de que se compruebe la primera posición o no.

5. Para el caso de prueba del anterior apartado, describe el primero de los estados erróneos. Describe detalladamente todo el estado (todas las variables, incluyendo el contador de programa).
El primer estado erróneo se produce cuando la variable i sólo toma valores desde el último hasta el segundo en sentido decreciente, cuando debería iterar hasta la primera posición.
6. Corrige el código y comprueba que todos los casos de prueba diseñados no provocan disfunciones en el programa.
Código corregido añadido al repositorio. Test corrido con éxito.

Lastzero.java

1. Explica el fallo (qué está mal en el código). Describe de manera precisa el fallo y propón una modificación al código que lo arregle.
El fallo se encuentra en la inicialización del for → for (int i = 0; i < x.length; i++) ya que comienza recorriendo por la primera posición del array hasta la última, devolviendo por tanto la posición del primer cero, y no del último.
Se podría corregir haciendo que la variable i comience en la última posición del array y vaya decreciendo hasta la primera, mientras busca un cero y devuelve su posición si lo encuentra. → for (int i=x.length-1; **i >=0**; i--).
2. Proporciona, si ello es posible, un caso de prueba que no ejecute el código que tiene el fallo. Si no es posible, explica por qué.
Java LastZero hace que no se ejecute el código de la función FindLast porque antes salta el manejador de error de lanzamiento del programa. A parte de este caso particular, el resto ejecutarán la función FindLast y por tanto se ejecutará el código donde se encuentra el fallo.
3. Si es posible, proporciona un caso de prueba que ejecuta el fallo que hay en el código, pero que no provoque un error en el estado. Si no se puede, explica por qué.
Ejemplo) java FindLast 1 2 3 0
Esto devolverá que la posición del último 0 es la 4. El programa no pasará por ningún estado erróneo ya que irá incrementando el valor de la variable i hasta que encuentre el 0 que está en la última posición.
4. Si es posible, proporciona un caso de prueba que provoque un error en el estado, pero que no acabe provocando una disfunción en el comportamiento del programa. No olvides que el contador de programa forma parte, junto a las variables, del estado del programa. Si no es posible, explica por qué.
No es posible pero se explicará algo similar con un ejemplo.
Ejemplo) java FindLast 1 2 3 0 6
El programa devolverá 3 como output, lo cual es correcto ya que la posición del último 0 es la 3 del aray. No se produce error como tal ya que todas las variables son consistentes, no se comprueba la última posición del array, lo cual es un error “potencial” pero no un error real de estado.
5. Para el caso de prueba del anterior apartado, describe el primero de los estados erróneos. Describe detalla- damente todo el estado (todas las variables, incluyendo el contador de programa).
No hay estado erróneo. Hay un fallo de implemanteación que conduce a “posibles estados erróneos” que no llegan a materializarse porque el programa finaliza al encontrar el primer cero. En realidad

el programa no tiene un error, la implementación sí. Esto hace que el programa no esté en estados erróneos, pero debido a la implementación no busca ceros más allá del primero encontrado por la izquierda.

6. Corrige el código y comprueba que todos los casos de prueba diseñados no provocan disfunciones en el programa.

Añadido al repositorio en GitHub.

CountPositive.java

1. Explica el fallo (qué está mal en el código). Describe de manera precisa el fallo y propón una modificación al código que lo arregle.

El fallo del código se encuentra en esta sentencia en la función CountPositive → if (x[i] >= 0) . El fallo se debe a que los ceros no se deben contar como positivos, por lo que sobra el igual.

El fallo se solucionaría cambiando esa línea por → if (x[i] > 0), es decir, quitando el igual.

2. Proporciona, si ello es posible, un caso de prueba que no ejecute el código que tiene el fallo. Si no es posible, explica por qué.

Java CountPositive hace que no se ejecute el código de la función CountPositive porque antes salta el manejador de error de lanzamiento del programa. A parte de este caso particular, el resto ejecutarán la función CountPositive y por tanto se ejecutará el código donde se encuentra el fallo.

3. Si es posible, proporciona un caso de prueba que ejecuta el fallo que hay en el código, pero que no provoque un error en el estado. Si no se puede, explica por qué.

Ejemplo) Java CountPositive -1 -2 -3

El código erróneo se ejecutará pero, dado que todos los número son negativos, la variable que cuenta los números positivos tendrá un estado correcto, al igual que el resto del programa.

Otro Ejemplo) Java CountPositive 1 2 3 6 -1

Dado que no hay ceros, se ejecutará el código con fallo pero el programa no pasará por un estado erróneo.

Todos los ejemplos en los que no se contenga un 0 son válidos en este caso.

4. Si es posible, proporciona un caso de prueba que provoque un error en el estado, pero que no acabe provocando una disfunción en el comportamiento del programa. No olvides que el contador de programa forma parte, junto a las variables, del estado del programa. Si no es posible, explica por qué.

No es posible dado que para que se provoque un estado erróneo tiene que haber un 0. De producirse este error, se acabará manifestando en la salida del programa → failure.

5. Para el caso de prueba del anterior apartado, describe el primero de los estados erróneos. Describe detalla- damente todo el estado (todas las variables, incluyendo el contador de programa).

No hay caso anterior.

6. Corrige el código y comprueba que todos los casos de prueba diseñados no provocan disfunciones en el programa.

Código subido al repositorio.

OddOrPos.java

1. Explica el fallo (qué está mal en el código). Describe de manera precisa el fallo y propón una modificación al código que lo arregle.

El fallo en el código se encuentra en la función OddOrPos, en la siguiente línea → if (x[i]%2 == 1 || x[i] > 0) debido a que si se realiza el módulo de un número negativo impar (-3 por ejemplo) el resultado no será 1 sino -1. Es decir, el programa inicial no contempla los números impares negativos. Para solucionarlo se debe hacer el valor absoluto de la operación módulo2 para que se contemplen así los impares negativos también → if (Math.abs(x[i]%2) == 1 || x[i] > 0).

2. Proporciona, si ello es posible, un caso de prueba que no ejecute el código que tiene el fallo. Si no es posible, explica por qué.

Java OddOrCero hace que no se ejecute el código de la función OddOrCero porque antes salta el manejador de error de lanzamiento del programa. A parte de este caso particular, el resto ejecutarán la función OddOrCero y por tanto se ejecutará el código donde se encuentra el fallo.

3. Si es posible, proporciona un caso de prueba que ejecuta el fallo que hay en el código, pero que no provoque un error en el estado. Si no se puede, explica por qué.

Cualquier ejemplo en el que se no se introducen números impares negativos hace que se ejecute el código erróneo pero no se produzca un error en el programa.

4. Si es posible, proporciona un caso de prueba que provoque un error en el estaod, pero que no acabe provocando una disfunción en el comportamiento del programa. No olvides que el contador de programa forma parte, junto a las variables, del estado del programa. Si no es posible, explica por qué.

No es posible debido a que desde el momento en el que se atraviesa un estado erróneo este se propaga hasta el output del programa, provocando un failure.

5. Para el caso de prueba del anterior apartado, describe el primero de los estados erróneos. Describe detalla- damente todo el estado (todas las variables, incluyendo el contador de programa).

No existe caso anterior.

6. Corrige el código y comprueba que todos los casos de prueba diseñados no provocan disfunciones en el programa.

Código incluído en el repositorio de GitHub.