Trazas y pruebas software

PPROG

Índice de contenidos

Introducción: Trazas versus pruebas

- Técnicas empleadas en el proceso de identificar errores en el software

- **Trazas:** permiten analizar la ejecución para detectar errores e incongruencias

- **Pruebas:** creación de escenarios que fuerzan situaciones específicas para comprobar que la ejecución es la deseada bajo ese escenario

Tipos de errores

- Sintácticos
  - De acuerdo a un lenguaje de programación
  - Triviales de detectar y corregir → El **compilador hace la detección** y avisa

- Estáticos
  - El código es sintácticamente correcto pero semánticamente incorrecto
    - P.e. cambio de “>” por “<”, “&&” por “||”, etc.

- Dinámicos
  - El código es sintáctica y semánticamente correcto, pero viola supuestos implícitos durante la ejecución
    - P.e. las variables toman valores no permitidos, se accede a posiciones de arrays que están fuera de rango, no se libera memoria correctamente, etc.

Una primera aproximación para hacer trazas de ejecución

- Insertar en el programa sentencias que impriman mensajes en momentos determinados.
  - P.e. para ver el valor de ciertas variables antes y después de la invocación a una función, para saber si se está en un camino de una bifurcación u otro, etc.

- ¿Dónde se envían los mensajes?
  - Salida estándar → stdout
    - `printf("Mensaje a mostrar");`
    - `fprintf(stdout, "Mensaje a mostrar");`
  - Salida estándar de error → stderr
    - `fprintf(stderr, "Mensaje a mostrar");`

- Puede ayudar a detectar errores estáticos y dinámicos

Una primera aproximación para hacer trazas de ejecución

- ¿Y si se insertan mensajes que no se desea que aparezcan en la ejecución final de la aplicación?

- **Solución:** dos versiones → una que genera mensajes y otra que no

- **¿Cómo?**
  - Compilación condicional → directivas de precompilación

```
#define DEBUG
…
#ifdef DEBUG
  printf("Some message");
#endif
```

Introducción a las pruebas software

- Construcción de un "conjunto de sentencias" que prueben la funcionalidad del software
  - Automatización

- Se tratará de "forzar" la ejecución de funcionamiento en:
  - Situaciones típicas
  - Situaciones extremas

Una primera aproximación a las pruebas software

```
ELEMBAG e1=3, e2=6, e3=9;
BAG b1, b2;

printElemBag (stdout, &e1); printf ("\n");
printElemBag (stdout, &e2); printf ("\n");
printElemBag (stdout, &e3); printf ("\n");

printf ("Inicializando bolsa...\n");
iniBag (&b1);
printBag (stdout, &b1);

printf ("Insertando elementos...\n");
insertElemBag (&b1, &e1);
printBag (stdout, &b1);

insertElemBag (&b1, &e2);
printBag (stdout, &b1);

insertElemBag (&b1, &e3);
printBag (stdout, &b1);

/* probamos a insertar en bolsa llena*/
if (insertElemBag (&b1, &e3) == ERROR)
   printf ("Bolsa 1 llena\n");
else
   printBag (stdout, &b1);
```

```
printf ("Copiando bolsa 1 en bolsa 2...\n");
copyBag (&b2, &b1);

printf ("Extrayendo elementos de bolsa
1...\n");

extractElemBag (&b1, &e1);
printBag (stdout, &b1);

extractElemBag (&b1, &e1);
printBag (stdout, &b1);

extractElemBag (&b1, &e1);
printBag (stdout, &b1);

/* probamos a extraer de bolsa vacía */
if (extractElemBag (&b1, &e1) == ERROR)
   printf ("Bolsa 1 vacia\n");
else
   printBag (stdout, &b1);

printf ("Estado de la bolsa 2:\n");
printBag (stdout, &b2);

printf ("Liberando bolsa 2...\n");
freeBag (&b2);
printf ("Estado de la bolsa 2:\n");
printBag (stdout, &b2);
```