Estructuras de Datos y Algoritmos

Tecnólogo en Informática

EVITANDO ERRORES: UTILIZACIÓN DE LA BIBLIOTECA <ASSERT.H>

INTRODUCCIÓN

La ideal es escribir código libre de errores, el problema es que en proyectos de mediano y gran porte esto no resulta simple. Por lo tanto, uno debe intentar detectar éstos en la etapa más temprana posible (es importante que sea antes de realizar la entrega).

¿Cuáles son las etapas en que se detectan errores y cuanto tiempo lleva repararlos?

- Cuando se está escribiendo el código, casi no lleva tiempo de reparación.
- Cuando se compila o linkedita, lleva unos segundos.
- Cuando el programa está iniciando, tal vez algunos minutos.
- Cuando se están ejecutando secuencias de testeo sobre el programa, puede llevar un buen rato repararlo.
- Cuando el programa ya fue entregado y lo están probando los docentes, esto no debe ocurrir.

Una manera de detectar errores es mediante la utilización de la biblioteca <assert.h> que provee el compilador.

FUNCIONAMIENTO

Éste es muy simple. Básicamente se provee de una macro llamada *assert* que se comporta como una función y permite evaluar una condición booleana para verificar que sea verdadera. En caso de que sea falsa el programa terminará imprimiendo un mensaje en la salida de errores, que incluye el nombre del fuente, el número de línea y el texto de la condición.

Pseudo-código de la macro *assert*:

```
void assert (bool condicion){
      if (! condicion){
            ImprimirError ("condición no satisfecha");
            TerminarPrograma ();
      }
}
```

De esta manera, la función "asegura" que la condición es satisfecha para continuar ejecutando el programa.

Se puede dejar sin efecto la función **assert** incluyendo la siguiente línea al principio del programa:

```
#define NDEBUG
```

o definiendo la constante mientras se compila:

```
gcc –c –D NDEBUG <archivo.c>
```

Como los asserts pueden ser sacados, no se debe incluir código con efectos secundarios en la condición, como por ejemplo pedir memoria.

UTILIZACIÓN

Se puede utilizar donde se desee verificar un condición de borde. Por ejemplo:

- Al comienzo de un procedimiento, verificando precondiciones.
- Al final de un procedimiento, verificando poscondiciones.
- Al pedir memoria, verificando que el sistema la está otorgando.

EJEMPLOS

1)

```
#include <assert.h>

list tail (list l){
      assert (! IsEmpty (l));
      return l->next;
}
```

2)

```
#include <assert.h>

list cons (list l, info i){
      list tmp;
      tmp = (list) malloc(sizeof(nodo));
      assert (tmp != NULL);
      tmp->info = i;
      tmp->next = l;
      return tmp;
}
```

No vale la pena verificar la memoria retornada por el operador **new** porque en caso de no poder otorgarla el programa termina automáticamente.