16. Controlling Loops

LOOP ”ES UN TÉRMINO INFORMAL que se refiere a cualquier tipo de estructura de control iterativo, cualquier estructura que haga que un programa ejecute repetidamente un bloque de código.

Los tipos de bucle comunes son for, while y do-while en C ++ y Java.

# 16.1 Selecting the Kind of Loop

## 16.1.1 When to Use a while Loop

● Los programadores novatos a veces piensan que un ciclo while se evalúa continuamente y que termina en el instante en que la condición while se vuelve falsa, sin importar qué declaración del ciclo se esté ejecutando (Curtis et al. 1986).

● Aunque no es tan flexible, un ciclo while es una opción de ciclo flexible. Si no sabe de antemano exactamente cuántas veces desea que se repita el ciclo, utilice un ciclo while.

## 16.1.2 When to Use a for Loop

● Un bucle for es una buena opción cuando necesita un bucle que se ejecute un número específico de veces. Puede usarlo en C ++, C, Java, Visual Basic y la mayoría de los otros lenguajes.

● Úselo para bucles para actividades simples que no requieren controles de bucle internos. Úselos cuando el control de bucle implica incrementos o decrementos simples.

● El punto de un bucle for es que lo configura en la parte superior del bucle y luego se olvida de él. No tienes que hacer nada dentro del bucle para controlarlo. Si tiene una condición bajo la cual la ejecución tiene que saltar de un bucle, use un bucle while en su lugar.

● Del mismo modo, no cambie explícitamente el valor del índice de un bucle for para forzar su finalización. En su lugar, use un bucle while. El bucle for es para usos simples. Las tareas de bucle más complicadas se manejan mejor con un bucle while.

## 16.1.3 When to Use a foreach Loop

● El bucle foreach o su equivalente (foreach en C #, For-Each en Visual Basic, For-In en Python) es útil para realizar una operación en cada miembro de una matriz u otro contenedor.

● Tiene la ventaja de eliminar la aritmética de mantenimiento de ciclos y, por lo tanto, eliminar cualquier posibilidad de errores en la aritmética de mantenimiento de ciclos.

# 16.2 Controlling the Loop

¿Qué puede salir mal con un bucle? Cualquier respuesta debería incluir al menos:

● inicialización de bucle incorrecta u omitida,

● inicialización omitida de acumuladores u otras variables relacionadas con el bucle,

● anidamiento inadecuado,

● terminación incorrecta del bucle,

● olvidar incrementar una variable de ciclo o incrementar la variable incorrectamente,

● e indexar incorrectamente un elemento de matriz de un índice de bucle.

Puede prevenir estos problemas observando dos prácticas:

1. Primero, minimice la cantidad de factores que afectan el bucle. ¡Simplificar! ¡Simplificar! ¡Simplificar!

2. En segundo lugar, trate el interior del ciclo como si fuera una rutina; mantenga tanto control como sea posible fuera del ciclo. Indique explícitamente las condiciones bajo las cuales se ejecutará el cuerpo del ciclo. No haga que el lector mire dentro del bucle para comprender el control del bucle. Piense en un bucle como una caja negra

C++ Example of Treating a Loop as a Black Box:

What are the conditions under which this loop terminates?:

while ( !inputFile.EndOfFile() && moreDataAvailable ) {

...

}

### Exiting Loops Early

● Muchos idiomas proporcionan un medio para hacer que un bucle finalice de alguna manera que no sea completar la condición for o while. En esta discusión, romper es un término genérico para romper en C ++, C y Java.

● La declaración de interrupción (o equivalente) hace que un bucle termine a través del canal de salida normal; el programa reanuda la ejecución en la primera instrucción que sigue al ciclo.

● Considere usar declaraciones de ruptura en lugar de indicadores booleanos en un ciclo while.

○ En algunos casos, agregar indicadores booleanos a un ciclo while para emular las salidas del cuerpo del ciclo dificulta la lectura del ciclo.

● A veces, puede eliminar varios niveles de sangría dentro de un bucle y simplificar el control del bucle con solo usar un descanso en lugar de una serie de pruebas if.

○ Poner varias condiciones de ruptura en declaraciones separadas y colocarlas cerca del código que produce la ruptura puede reducir el anidamiento y hacer que el bucle sea más legible.

● Tenga cuidado con un bucle con muchas rupturas dispersas:

○ Un bucle que contiene muchas interrupciones puede indicar un pensamiento poco claro sobre la estructura del bucle o su función en el código circundante. Una proliferación de descansos

plantea la posibilidad de que el bucle pueda expresarse más claramente como una serie de bucles en lugar de como un bucle con muchas salidas.

● Utilice rotura etiquetada si su idioma lo admite.

○ Java admite el uso de descansos etiquetados

● Use pausa y continúe solo con precaución

○ El uso de break elimina la posibilidad de tratar un bucle como una caja negra. Limitarse a una sola declaración para controlar la condición de salida de un bucle es una forma poderosa de simplificar sus bucles. El uso de una pausa obliga a la persona que lee su código a mirar dentro del bucle para comprender el control del bucle. Eso hace que el bucle sea más difícil de entender.

○ Algunos científicos informáticos argumentan que son una técnica legítima en la programación estructurada; algunos argumentan que no lo son. Debido a que no sabe en general si continuar y romper es correcto o incorrecto, utilícelos, pero solo con el temor de estar equivocado.

● Comprobación de puntos finales

○ Un solo bucle suele tener tres casos de interés: el primer caso, un caso intermedio seleccionado arbitrariamente y el último caso. Cuando crea un bucle, recorra mentalmente el primer, el medio y el último caso para asegurarse de que el bucle no tenga errores uno por uno. Puede esperar varios beneficios de las simulaciones mentales y los cálculos manuales.

○ La disciplina mental da como resultado menos errores durante la codificación inicial, una detección más rápida de errores durante la depuración y una mejor comprensión general del programa. El ejercicio mental significa que comprendes cómo funciona tu código en lugar de adivinarlo.

### Using Loop Variables

Aquí hay algunas pautas para usar variables de bucle:

● Utilice tipos ordinales o enumerados para límites tanto en matrices como en bucles.

Generalmente, los contadores de bucle deben ser valores enteros.

● Utilice nombres de variables significativos para hacer que los bucles anidados sean legibles.

○ Las matrices a menudo se indexan con las mismas variables que se utilizan para los índices de bucle.

○ Si tiene una matriz unidimensional, es posible que pueda usar i, j o k para indexarla.

○ Pero si tiene una matriz con dos o más dimensiones, debe usar nombres de índice significativos para aclarar lo que está haciendo.

● Limitar el alcance de las variables de índice de bucle al bucle en sí.

La diafonía de índice de bucle y otros usos de los índices de bucle fuera de sus bucles es tal un problema importante.

### How Long Should a Loop Be?

La longitud del bucle se puede medir en líneas de código o profundidad de anidamiento.

● Haz que tus bucles sean lo suficientemente cortos para verlos todos a la vez.

Sin embargo, cuando empiece a apreciar el principio de escribir código simple, rara vez escribirá bucles de más de 15 o 20 líneas.

● Limite el anidamiento a tres niveles

○ Los estudios han demostrado que la capacidad de los programadores para comprender un bucle se deteriora significativamente más allá de los tres niveles de anidación (Yourdon 1986a).

○ Si va más allá de esa cantidad de niveles, acorte el ciclo (conceptualmente) dividiendo parte de él en una rutina o simplificando la estructura de control.

● Mover las entrañas de los bucles de los bucles largos en rutinas (mover las entrañas de un loop extenso a rutinas):

Si el ciclo está bien diseñado, el código en el interior de un ciclo a menudo se puede mover a una o más rutinas que se llaman desde dentro del ciclo.

● Haga que los bucles largos sean especialmente claros: la longitud agrega complejidad.

○ Si escribe un bucle corto, puede utilizar estructuras de control más riesgosas como romper y continuar, salidas múltiples, condiciones de terminación complicadas, etc.

○ Si escribe un bucle más largo y siente alguna preocupación por su lector, le dará al bucle una única salida y dejará la condición de salida inequívocamente clara.

# 16.3 Creating Loops Easily—from the Inside Out

EJEMPLIFICARLO con el ejercicio de la TABLA DE MULTIPLICAR.

Este es el proceso general:

1. Comience con un caso. Codifique ese caso con literales.

2. Luego, aplique una sangría y reemplace los literales con índices de bucle o expresiones calculadas.

3. Ponga un lazo alrededor.

4. Pon otro bucle alrededor de eso, si es necesario, y reemplaza más literales.

5. Continúe el proceso todo el tiempo que sea necesario.

6. Cuando termine, agregue todas las inicializaciones necesarias.

7. Si ha usado un while, añade la sentencia que modifica la expresión de entrada al while.

8. Dado que comienza con el caso simple y trabaja hacia afuera para generalizarlo, podría pensar en esto como una codificación de adentro hacia afuera.

# 16.4 Correspondence Between Loops and Arrays

Los bucles y las matrices suelen estar relacionados. En muchos casos, se crea un bucle para realizar una manipulación de la matriz, y los contadores de bucles se corresponden uno a uno con los índices de la matriz. Por ejemplo, los índices de bucle for de Java a continuación corresponden a los índices de la matriz:

Algunos lenguajes proporcionan potentes operaciones de matriz que eliminan la necesidad de bucles como el anterior. Aquí hay un fragmento de código APL que realiza la misma operación:

APL Example of an Array Multiplication

Product <- a x b

The language you use to solve a problem substantially affects your solution.

# Key Points

● Los bucles son complicados. Mantenerlos simples ayuda a los lectores de su código.

● Las técnicas para mantener los bucles simples incluyen evitar tipos exóticos de bucles, minimizar el anidamiento, hacer que las entradas y salidas sean claras y mantener el código de limpieza en un solo lugar.

● Los índices de bucle están sujetos a un gran abuso. Nómbrelos claramente y utilícelos para un solo propósito.

● Piense detenidamente en el ciclo para verificar que funciona normalmente en cada caso y termina en todas las condiciones posibles.

# CHECKLIST: Loops

Selección y creación de bucles

❏ ¿Se utiliza un ciclo while en lugar de un ciclo for, si corresponde?

❏ ¿Se creó el bucle de adentro hacia afuera?

Entrar en el bucle

❏ ¿Se ingresa al bucle desde arriba?

❏ ¿El código de inicialización está directamente antes del bucle?

❏ Si el bucle es un bucle infinito o un bucle de eventos, ¿está construido limpiamente en lugar de usar un kludge como para i = 1 a 9999?

❏ Si el bucle es un bucle for de C ++, C o Java, ¿el encabezado del bucle está reservado para el código de control de bucle?

Dentro del Loop

❏ ¿El bucle utiliza {y} o su equivalente para evitar problemas derivados de modificaciones incorrectas?

❏ ¿El cuerpo del lazo tiene algo dentro? ¿No está vacío?

❏ ¿Se agrupan las tareas domésticas, ya sea al principio o al final del ciclo?

❏ ¿El ciclo realiza una y solo una función, como lo hace una rutina bien definida?

❏ ¿Es el bucle lo suficientemente corto para verlo todo a la vez?

❏ ¿El circuito está anidado en tres niveles o menos?

❏ ¿Se han trasladado los contenidos del ciclo largo a su propia rutina?

❏ Si el bucle es largo, ¿está especialmente claro?

Índices de bucle

❏ Si el bucle es un bucle for, ¿el código dentro de él evita los monos con el índice del bucle?

❏ ¿Se utiliza una variable para guardar valores de índice de ciclo importantes en lugar de utilizar el índice de ciclo fuera del ciclo?

❏ ¿El índice de bucle es de tipo ordinal o enumerado, no de punto flotante?

❏ ¿Tiene el índice de bucle un nombre significativo?

❏ ¿El bucle evita la interferencia de índices?

Saliendo del Loop

❏ ¿Termina el bucle en todas las condiciones posibles?

❏ ¿El circuito utiliza contadores de seguridad, si ha instituido un estándar de contadores de seguridad?

❏ ¿Es obvia la condición de terminación del bucle?

❏ Si se utilizan romper o continuar, ¿son correctos?