aprenderaprogramar.com

Ejercicio ejemplo resuelto arrays (arreglos) dinámicos II: opciones de variantes en el algoritmo (CU00215A)

Sección: Cursos

Categoría: Curso Bases de la programación Nivel II

Fecha revisión: 2024

Autor: Mario R. Rancel

Resumen: Entrega nº14 del Curso Bases de la programación Nivel II

24

Continuamos comentando el ejercicio realizado. Algunas opciones más pueden ser:

a) Calculamos el número de iteraciones si es posible, y en base a ello dimensionamos el vector. En nuestro caso sería:

```
3. Impar = – 1 : i = 0
4. Redimensionar Valor(Redondear((Dato / 2) + 0,1))    [+ 0,1 fuerza el redondeo al alza]
5. Hacer
        .
        .
        .
```

También:

```
3. Impar = – 1 : i = 0
4. Redimensionar Valor((Dato +1) / 2)
5. Hacer
        .
        .
        .
```

Al tener una dimensión exacta ya no será necesario hacer un redimensionamiento final para ajuste de datos válidos, ya que todos los datos serán válidos. Este método parece ideal y quizás lo sea. El problema está en que no siempre vamos a conocer el número de iteraciones al haber muchas situaciones en que es indefinido a priori. En esta coyuntura resulta útil la asignación por paquetes.

b) Asignación por paquetes o grupos. Si tenemos un rango de variación de entre 1 y 1000 datos, puede ser interesante no usar un array de 1000 localizadores que no hay que ampliar pero es exageradamente grande. Tampoco conviene un array de un solo elemento que hay que ampliar constantemente. La solución puede ser intermedia: ir cogiendo paquetes de datos de 20 en 20, de 30 en 30, etc. que vamos rellenando. Si el array se llena, se amplía en otro grupo de 20 ó 30 y si se acaba el proceso, ajustamos el tamaño final al de datos reales sin haber usado un array desproporcionadamente grande. En nuestro caso sería:

```
3. Impar = – 1 : i = 0
4. Redimensionar Valor(20)
5. Hacer
        i = i + 1
        Si i = Limitesuperior(Valor) + 1 Entonces
            Redimensionar Valor(i + 19)
        FinSi
        Impar = Impar + 2
        Valor(i) = Impar
   Repetir Mientras Impar < Dato
6. Redimensionar Valor(i)
7. Cantidad = i
```

c) Realizar un reconocimiento inicial del número de datos para después redimensionar y realizar la asignación. En este ejemplo vendría siendo algo del tipo:

3. Impar = – 1 : i = 0
4. **Hacer**

   i = i + 1

   Impar = Impar + 2

   Repetir Mientras Impar < Dato
5. Redimensionar Valor(i)
6. Impar = – 1 : i = 0
7. **Hacer**

   i = i +1

   Impar = Impar + 2

   Valor(i) = Impar

   Repetir Mientras Impar < Dato
8. Cantidad = i

El tener que realizar un recorrido inicial por todos los datos y otro recorrido de asignación puede ser altamente ineficiente, sobre todo cuando el número de datos es grande. Por tanto, existiendo otras alternativas esta carecerá de interés por no interesarnos duplicar un proceso innecesariamente, con el consumo de tiempo y recursos que llevará aparejado.

Visto esto, comentaremos ahora el por qué se limita la entrada de datos a 1 – 19. Tenemos que remitirnos a lo indicado cuando hablamos de la declaración de variables: los ordenadores tienen mucha potencia y capacidad, pero no infinita. Y hemos de atender tanto a los datos de entrada como a los de salida.

Un rango de entrada entre 1 y 19 parece que es escaso. Pero si analizamos las salidas previsibles nos encontraríamos que los valores del producto $1 * 3 * 5 * \cdots * n$ son:

Para n = 1 → 1

Para n = 5 → 15

Para n = 9 → 945

Para n = 15 → 2027025

Para n = 19 → 654729075

Para n = 29 → $6{,}190283354 \cdot 10^{15}$

Trabajar con números de muy "grueso calibre" terminará por desbordar la capacidad del ordenador si no los mantenemos controlados.

Otro aspecto por el que se pueden acotar los datos de entrada es el interés del programador. Si un dato de entrada corresponde al número de trabajadores de una empresa. ¿Para qué crear un algoritmo que funcione bien cuando el dato es negativo si nunca va a ser negativo? En casos como éste acotamos el problema y nos centramos en que funcione bien para las circunstancias concretas a que nos enfrentamos. ¿Está esto en contradicción con la recomendación de resolver siempre problemas genéricos? No, una cuestión es tratar de resolver supuestos generales y otra tratar de resolverlo todo. Por otro lado hay que tener en cuenta que muchas veces para amplificar el campo de acción de un algoritmo complejo lo mejor puede ser empezar resolviendo determinadas situaciones (p. ej. números positivos pequeños) para después, a través de pequeñas adaptaciones, llegar al algoritmo de resolución general.

Pasemos a otra cuestión. En el programa tenemos variables globales y variables locales. Observamos, por un lado, que la variable *i* aparece repetida como variable local en distintos módulos. ¿Es esto posible? Sí, porque la variable local es conocida en su módulo. No es demasiado aconsejable declarar la misma variable en distintos puntos del programa porque puede inducir a confusión. En el caso de *i* no tiene demasiada importancia porque únicamente actúa como contador a nivel local, no siendo portadora de información.

¿Podría haberse declarado *i* también como variable global? No, deberá ser global o local pero evitaremos que se den ambas circunstancias por no ser necesario y por impedir el correcto control del programa.

Observamos que en el módulo *Genera* se declara en la línea 1 *Dato = 0*. Esto es necesario para acceder al bucle de entrada de datos inicial. Si no lo hiciéramos, *Dato* conservaría su valor anterior y no se entraría en la petición del nuevo dato. *Dato* es una variable global: se inicia a valor nulo cada vez que se corre el programa. Por el contrario, las variables locales se inician cada vez que es llamado el módulo donde están declaradas. Así, en el módulo *CalculaSuma* la declaración en la línea 1 de *Suma = 0* es redundante. Los programadores a veces usan este tipo de declaraciones para:

- "Recordarse" que el módulo empieza con *Suma* valiendo cero.
- Pasar de un lenguaje que no requiere estas declaraciones a otro que sí las requiera sin tener que estar revisando la asignación inicial de valores a variables.

En cambio la variable *Acumular* es necesario establecerla con valor inicial igual a 1 para que funcione.

Otra cuestión que observamos es que en el módulo *CalculaSuma* el parámetro requerido se llama *Numero* y en el módulo *CalculaMult* tiene por nombre *Cifra.* El nombre de dos parámetros en distintos módulos puede coincidir pues tienen carácter local, pero no es muy recomendable ya que en vez de independizar los módulos lleva a que se confundan entre ellos. La llamada a los módulos se ha hecho por valor, que según hemos dicho es más seguro, aunque en este caso no tiene trascendencia el que se llame por valor o por variable.

Próxima entrega: CU00216A

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:
http://www.aprenderaprogramar.com/index.php?option=com_content&view=category&id=36&Itemid=60