Programación Genérica y Patrones de Diseño de Software

Tarea No. 3

Tomás Balderas Contreras
balderas@ccc.inaoep.mx
Coordinación de Ciencias de la Computación
Instituto Nacional de Astrofísica, Óptica y Electrónica

4 de julio de 2003

1 Ejercicios: Capítulo 4

1. Definir el constructor de la clase `Matrix<T>`, cuya definición se encuentra en el programa 4.22, página 91 de [1].

   Solución:

   La clase `Matrix<T>` hereda las tres variables instancia y todos los métodos de la clase `Array2D<T>`, entre ellos el constructor `Array2D<T>::Array2D(unsigned int, unsigned int)`. La clase `Matrix<T>` no define otras variables instancia y, por lo tanto, para inicializar esta clase basta que el constructor `Matrix<T>::Matrix(unsigned int, unsigned int)` invoque al constructor de su clase base (`Array2D<T>`) proporcionándole sus propios parámetros. El código del constructor es el siguiente:

   ```
   template <class T> Matrix<T>::Matrix(unsigned int m, unsigned int n) :
       Array2D<T>(m, n) {
   }
   ```

2. Sobrecargar el operador `+` en la clase `Matrix<T>`.

   Solución:

   El autor del libro no proporciona el código del método que sobrecarga el operador `+` para la clase `Matrix<T>`. El siguiente listado de código muestra la implantación de tal método:

   ```
   template <class T> Matrix<T> Matrix<T>::operator+(Matrix<T>& arg) {
       if (numberOfRows != arg.numberOfRows)
           throw invalid_argument("incompatible matrices");
       if (numberOfColumns != arg.numberOfColumns)
           throw invalid_argument("incompatible matrices");

       Matrix<T> result(numberOfRows, numberOfColumns);
       for (unsigned int i = 0; i < numberOfRows; ++i)
   ```

```
        for (unsigned int j = 0; j < numberOfColumns; ++j)
            result[i][j] = (*this)[i][j] + arg[i][j];
    return result;
}
```

3. La clase `Array2D<T>` declarada en el programa 4.20, página 88, sólo permite intervalos que comienzan en cero para los índices. Modificar la implantación para permitir un valor base para los índices en cada dimensión.

   Solución:

   Primero es necesario agregar dos variables instancia a la definición de la clase, cada una de estas variables es un valor de tipo `int` y contiene el valor inicial que pueden tomar los índices en el arreglo. Este valor inicial puede ser positivo o negativo. El código de la nueva clase es el siguiente:

```
template <class T> class Array2D {
 protected:
    int           rowBase;
    unsigned int  numberOfRows;
    int           columnBase;
    unsigned int  numberOfColumns;
    Array<T>      array;
 public:
    class Row {
        Array2D&   array2D;
        int const  row;
     public:
        Row(Array2D& _array2D, int _row) :
            array2D(_array2D),
            row(_row) {
        }
        T& operator[](int column) const {
            return array2D.Select(row, column);
        }
    };

    Array2D(unsigned int, int, unsigned int, int);
    T& Select(int, int);
    Row operator[](int);
};
```

   Notar que el constructor de la clase ahora necesita dos nuevos parámetros para asignar a las variables instancia `rowBase` y `columnBase`. El código para el constructor es el siguiente:

```
template <class T> Array2D<T>::Array2D(unsigned int m, int p,
                                       unsigned int n, int q) :
    rowBase(p),
    numberOfRows(m),
    columnBase(q),
    numberOfColumns(n),
```

```
    array(m * n) {
}
```

Otra diferencia con respecto a la clase `Array2D<T>` definida por el autor es que ahora los índices pueden tomar valores negativos y, por lo tanto, es necesario cambiar el tipo de las siguientes variables instancia y parámetros de métodos de `unsigned int` a `int`:

- la variable instancia `row` de la clase `Array2D<T>::Row`,
- el parámetro `_row` del constructor de la clase `Array2D<T>::Row`,
- el parámetro `column` en el método `Array2D<T>::Row operator[]`,
- los dos parámetros del método `Array2D<T>::Select` y
- el parámetro del método `Array2D<T>::operator[]`

En lo que respecta a la implantación de los métodos de la clase, la función miembro `Array2D<T>::operator[]` no sufre modificaciones en su cuerpo. Sin embargo, sí es necesario realizar cambios en el cuerpo del método `Array2D<T>::Select`. Primero, verificar que los índices i, j que recibe como argumentos son válidos, es decir, que cumplan con las siguientes desigualdades:

$$rowBase \leq |i| < rowBase + numberOfRows$$

$$columnBase \leq |j| < columnBase + numberOfColumns$$

Segundo, el polinomio de direccionamiento que calcula el desplazamiento en el arreglo lineal de un elemento i, j del arreglo de dos dimensiones debe involucrar los valores `rowBase` y `columnBase`. El código del método con los cambios sugeridos se muestra a continuación:

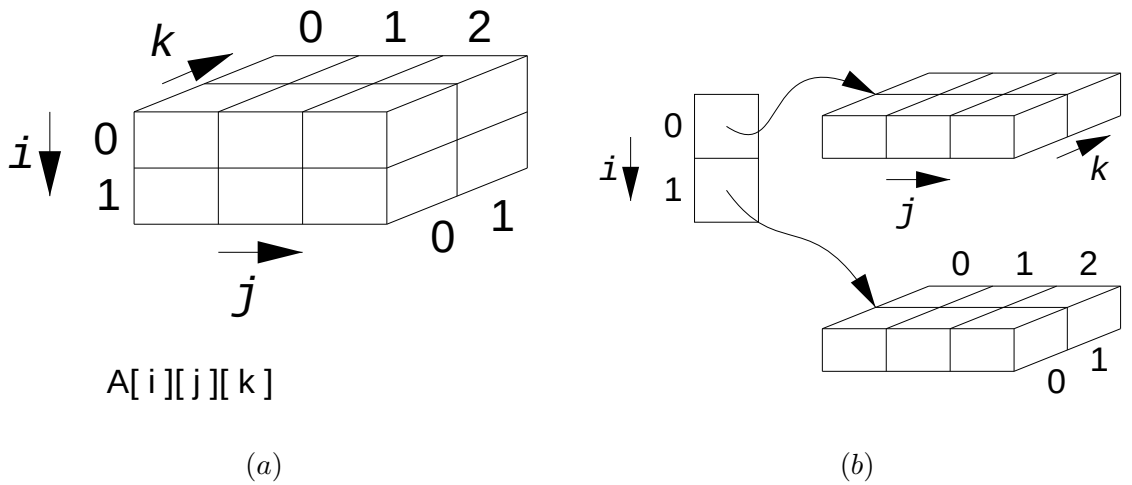
```
template <class T> T& Array2D<T>::Select(int i, int j) {
    if ( (i < rowBase) || (i >= rowBase + (int)numberOfRows) )
        throw out_of_range("invalid row");
    if ( (j < columnBase) || (j >= columnBase + (int)numberOfColumns) )
        throw out_of_range("invalid column");

    return array[(i - rowBase) * numberOfColumns + j - columnBase];
}
```

4. Diseñar e implantar una clase `Array3D<T>` para un arreglo de tres dimensiones en base a la clase `Array2D<T>`.

   Solución:

   Considerar un arreglo de tres dimensiones $A_{m \times n \times p}$ con n filas, p columnas y m "capas". La figura 1–(a) ilustra tal arreglo con $m = 2$, $n = 3$ y $p = 2$, donde el índice i recorre las capas, el índice j las filas y el índice k las columnas. El problema se resuelve si consideramos a un arreglo de tres dimensiones como un arreglo lineal de arreglos de dos dimensiones. La figura 1–(b) muestra el arreglo lineal construído a partir del arreglo de tres dimensiones A, cada elemento del arreglo lineal es un apuntador a una capa (arreglo de dos dimensiones) del arreglo tridimensional original.



(a) (b)

Figura 1: Un arreglo de tres dimensiones y su interpretación como un arreglo de arreglos.

En este momento contamos con una clase que define completamente arreglos de dos dimensiones (`Array2D<T>`) y en tiempo de ejecución podemos emplear el operador `new` para construir un arreglo lineal de instancias de esta clase. Por lo tanto, si encapsulamos este arreglo junto a tres variables enteras que indiquen las dimensiones del arreglo, un constructor y un método de acceso al i–ésimo elemento en el arreglo lineal, llegamos a la siguiente definición para la clase:

```
template <class T> class Array3D {
 protected:
    unsigned int        numberOfLayers;
    unsigned int        numberOfRows;
    unsigned int        numberOfColumns;
    Array2D<T>          *array;
 public:
    Array3D(unsigned int, unsigned int, unsigned int);
    Array2D<T>& operator[](unsigned int);
};
```

El constructor asigna valores iniciales a las variables que indican las dimensiones del arreglo tridimensional. La longitud del arreglo lineal que el constructor debe reservar está dada por la variable miembro `numberOfLayers` y cada instancia de `Array2D<T>` creada debe tener dimensiones `numberOfRows` $\times$ `numberOfColumns`, por lo tanto es necesario especificar que el constructor de la clase `Array2D<T>` debe ser invocado para cada instancia creada. El código del constructor es el siguiente:

```
template <class T> Array3D<T>::Array3D(unsigned int m,
                                       unsigned int n,
                                       unsigned int p) :
    numberOfLayers(m),
    numberOfRows(n),
```

```
    numberOfColumns(p) {
    array = new Array2D<T>[numberOfLayers](numberOfRows, numberOfColumns);
}
```

Cada elemento del arreglo puede ser alcanzado empleando tres índices i, j, k. Observar que el índice i especifica en qué capa se encuentra el dato, o bien, en qué objeto del arreglo lineal se encuentra el elemento. Una vez que tenemos una referencia a la capa donde está el dato, simplemente lo buscamos empleando los índices j y k. El método que sobrecarga el operador `[ ]` regresa una referencia a un objeto dentro del arreglo lineal y su código es el siguiente:

```
template <class T> Array2D<T>& Array3D<T>::operator[](unsigned int i) {
    return array[i];
}
```

De acuerdo a las descripciones de las clases con las que contamos podemos identificar los tipos de objetos involucrados en una sentencia de asignación a un elemento del arreglo tridimensional. El siguiente es un esquema de las instancias que intervienen y la clase a la que pertenecen:

$$\underbrace{\underbrace{\overbrace{\texttt{A}}^{\texttt{Array3D<T>}}\texttt{[i]}}_{\texttt{Array2D<T>\&}}\texttt{[j]}}_{\texttt{Array2D<T>::Row}}\texttt{[k]} = \ldots;$$

$$\texttt{T\&}$$

Referencias

[1] Preiss, B. R. *Data Structures and Algorithms With Object Oriented Design Patterns in C++*.