storage-mapping function. Consider the following example skeletal Fortran subroutine:

```fortran
Subroutine Sub(Matrix, Rows, Cols, Result
    Integer, Intent(In) :: Rows, Cols
    Real, Dimension(Rows, Cols), Intent(In) :: Matrix
    Real, Intent(In) :: Result

End Subroutine Sub
```

This works perfectly as long as the `Rows` actual parameter has the value used for the number of rows in the definition of the passed matrix. The number of rows is needed because Fortran stores arrays in column major order. If the array to be passed is not currently filled with useful data to the defined size, then both the defined index sizes and the filled index sizes can be passed to the subprogram. Then the defined sizes are used in the local declaration of the array, and the filled index sizes are used to control the computation in which the array elements are referenced. For example, consider the following Fortran subprogram:

```fortran
Subroutine Matsum(Matrix, Rows, Cols, Filled_Rows,
    Filled_Cols, Sum)
    Real, Dimension(Rows, Cols), Intent(In) :: Matrix
    Integer, Intent(In) :: Rows, Cols, Filled_Rows,
        Filled_Cols
    Real, Intent(Out) :: Sum
    Integer :: Row_Index, Col_Index
    Sum = 0.0
    Do Row_Index = 1, Filled_Rows
        Do Col_Index = 1, Filled_Cols
            Sum = Sum + Matrix(Row_Index, Col_Index)
        End Do
    End Do
End Subroutine Matsum
```

Java and C# use a technique for passing multidimensional arrays as parameters that is similar to that of Ada. In Java and C#, arrays are objects. They are all single-dimensional, but the elements can be arrays. Each array inherits a named constant (`length` in Java and `Length` in C#) that is set to the length of the array when the array object is created. The formal parameter for a matrix appears with two sets of empty brackets, as in the Java method below that does what the Ada example function `Sum` does:

```java
float sum(float mat[][][]) {
    float sum = 0.0f;
    for (int row = 0; row < mat.length; row++) {
        for (int col = 0; col < mat[0].length; col++) {
            sum += mat[row][col];
        }
    }
    return sum;
}
```