重返陣列：佈局附註、下標指定及多維度索引運算

Arrays Revisited: Layout Annotation, Subscript Assignment, and Multidimensional Index

吳培基

國立澎湖海事管理專科學校資訊工程科

澎湖縣880馬公市六合路300號
Email: pcwu@npit.edu.t

摘 要

陣列是電腦程式語言中重要的資料型別。陣列雖然基本但不同語言間也有許多襲用經年但並不相容的差異。陣列繫結方式分為編譯時期、物件產生時期及物件操作時期，分別稱為靜態、動態及彈性。陣列的佈局分為列為主及行為主二種，下標運算則有取得、指定及刪除等。本文提出下列有關陣列的程式架構：1)陣列宣告時可附加說明陣列的繫結及佈局方式。2)一種物件的下標運算的介面，恰包含下標取得及指定二類運算。以C++語法表示，分別為operator[]及operator[]。3)將下標運算推廣到多維度。這些語言架構可使陣列的介面更安全及更具威力。

關鍵詞：C++，動態繫結，靜態繫結，列為主，行為主。

Abstract

Arrays are fundamental and primitive data types in programming languages. Although primitive, arrays have many variations, which have been used for decades and are unfortunately incompatible with each other. There are three kinds of binding for arrays: compile-time, creation-time, and manipulation-time. These three are called static, dynamic, and flexible binding, respectively. There are also two kinds of layout order for arrays, row-major and column-major, and various subscript operators, such as get, assign, and delete. This paper presents the following language features for arrays: 1) An array declaration can be annotated with binding and layout order of the array. 2) An object interface consists of two subscript operators: get and assign. In C++ syntax, these are defined as operator '[]' and operator '[]='. 3) We extend subscript operators to be multidimensional. These language features make array interfaces safe and powerful.

Keywords: C++, dynamic binding, static binding, row-major, column-major.

1. INTRODUCTION

Arrays are fundamental and primitive data types in programming languages. Although primitive, arrays have many variations, which have been used for decades and are unfortunately incompatible with each other. Ghezzi and Jazayeri [5, p.100] introduce three kinds of binding for arrays: compile-time, object creation-time, and object manipulation-time. In the following, these three are called static, dynamic, and flexible binding, respectively. Arrays are static by default in languages such as FORTRAN [1], Pascal [7], C [8], and C++ [3]; arrays are dynamic in Java [4] and Eiffel [10]; Ada [2] provides both static and dynamic arrays; arrays in Smalltalk [6] and Python [9] are flexible. Algol 68 [5, p.101] provides both dynamic and flexible arrays, using the keyword 'flex' to indicate flexible arrays; otherwise, arrays are dynamic.

Another remarkable difference between arrays in various programming languages is the layout order for arrays. There are two conventions: row-major and column-major. C/C++ adopts row-major; BASIC and FORTRAN 77 adopt column-major. FORTRAN 90 does not insist on one specific order. When programs of different programming languages are linked together, the difference in the layout order of multidimensional arrays must be considered.

Subscripts are a fundamental programming feature for accessing elements of arrays. The C++ language allows subscript operators to be used in user-defined data types (i.e., classes). For example:

```
class a_class_name
{ ...
      T operator[] (S index);   ...
};
```

T is the return type; *S* is the type of index; '*a*[*i*]' denotes the element of the array (or object) *a* indicated by an index *i*. For *a*[*i*] to be used in the both sides of an assignment statement in C++, *T* is usually a reference data type, such as *char&* in the following:

```
class String
{ ...
      char& operator[] (int index);   ...
};
String a = "a test string";
a[3] = a[5];
```

The disadvantage is that this technique uses reference types, which ma be unavailable to some programming languages.

This paper presents the following language features for arrays: 1) Storage layout annotation: An array declaration can be annotated with binding and layout order of the array. 2) Subscript assignment: An object interface consists of two subscript operators: get and assign. In C++ syntax, these are operators '[]' and '[]='. 3) Multidimensional index: We extend subscript operators to be multidimensional. A two-dimensional index of subscript 'get' operator can be defined as operator [] (int x, int y). These language features make array interfaces safe and powerful.

2. Related Work

Flexible arrays are easy to use; however, their implementation needs data structures such as hash tables, which take more time and space than static and dynamic arrays. Thus, flexible arrays are not suitable used as the sole data type of arrays provided in a programming language, when the execution and space efficiency is considered crucial. Most recent programming languages such as Java and Eiffel provide only dynamic arrays. This may indicate that dynamic arrays are in major stream. However, C/C++ are also influential languages today, the chance using static arrays is still very high. Ada's data type declaration can be either static or dynamic, they are called constrained and unconstrained, respectively. The following are array declarations in Ada:

```
type GAME_BOARD is array (1..8, 1..8) of
CHESS_PIECE;
type BIT_VECTOR is array (INTEGER range <>)
of BOOLEAN;
```

GAME_BOARD is a data type of static arrays; BIT_VECTOR is a data type of dynamic arrays. Static and dynamic arrays are different data types, which cannot be mixed together.

Java does not allow an array to export the reference of its element. This is a simplification from C++, although Java inherits many features from C and C++. To be kept simple, Java even removes all C++ overloaded operators. The subscript operato does not apply to Java's user -defined classes. Fo example, when manipulating a *StringBuffer* [4, p.323] object, Java uses methods such as *charAt*, *setCharAt* to access characters in the object:

```
public final class StringBuffer extends Object
{     ...
      public char charAt(int index);
      public void setCharAt(int index, char ch);
}
```

Python provides rich subscript operators. Fo example, a partial list of the object methods of mutable sequence [9, p.787] are as follows:

Operation	Description	Object Method
s[i]	Index	__getitem__
s[i:j]	Slicing	__getslice__
s[i] = x	Index assignment	__setitem__
s[i:j] = x	Slice assignment	__setslice__
Del s[i]	Index deletion	__delitem__
Del s[i;j]	Slice deletion	__delslice__

Python's subscript operators can be classified into three categories: *get*, *assi* , and *delete.* Operati ns in Column 1 are mapped into calls of object methods in Column 3. Thus, one does not need to call *s.__getitem__(i)* directly but uses *s[i]* instead. Deleting elements by an index (or slice) is to remove data of the index (or the slice) in the series. Python's slice *s[i:j]* denotes accessing the contents of *s[i]* to *s[j-1]*.

There is another convention for slices. Slices *s[i:j]* in FORTRAN are defined as follows: *i* is the starting, *j* is the ending, and *s[i:j]* denotes accessing the contents of *s[i]* to *s[j]*. Slices can also contain disjoint elements. FORTRAN 90 [1] defines a subscript triplet as follows:

s[first : last : stride]

denotes one element for every stride from *s[first]* to *s[last]*.

3. Storage Layout

Foll wing the adoption of recent programming languages such as Java and Eiffel, dynamic binding for array data types becomes more widely used. However, this trend makes linking programs in different languages more difficult. To solve this difficulty, the decla ation of arrays can be annotated with the way of binding (static or dynamic), in addition with storage layouts of arrays. For example, the following annotation extends the syntax of C and C++:

```
double x[100] /dynamic ;
chess_piece board[8][8] /dynamic;
int a[] /static = {1, 2, 3, 4, 5} ;
```

```
double m[250][250] /dynamic /col-major ;
double t[4][4] /static /ro  -major;
double m[31] /dynamic;
```

The way of binding is categorized into static (/static) and dynamic (/dynamic). Storage layout can be categorized i nto row-major (/row-major) and column-major (/col-major). Storage layouts are only applicable to arrays of more than two dimensions. The way of binding is usually specified explicitl when the program is linked with programs of othe languages. Programming languages can provide a default binding for arrays. For example, the default of C and C++ can be treated as "/static /ro -major". Here the prefix of annotated keywords is '/', this can avoid the confusion with existing keywords, such as the keyword 'stati c' of C and C++. Using the syntax of Pascal to declare arrays is as follows:

```
var
x : /dynamic array [1..100] of real;
board : /dynamic array [1..8, 1..8] of chess_piece;
```

When arrays are used in the parameters of a function, they can also be annotated in their parameter declarations. For example, the following is a function *inverse*, which takes a static arra *a* wit *m* rows and *n* columns:

```
void inverse(double a[m][n] /static);
```

When a program is mixed with static and dynamic arrays, the system can autom aticall converts dynamic arrays into static arrays of equal dimension. Dynamic arrays usually have the following data structure:

```
struct dynamic_arra
{
    int n_dims;             // number of dimensions
    int length[n_dims] /static ;
        // length of each dimension
    T data[size] /static;
        // size = product of length[i]
};
```

Constant *n_dims* is the dimension of the array *length* is the size of each dimension; *T* is the data type of array elements; *data* is a static array fo storing each element of the array. Let *a0* be a dynamic array with *m* rows and *n* columns:

```
double a0[][] = new double[m][n] /dynamic;
```

Since the dimension of *a0* is equal to that of parameter *a* in the inverse function, we can use a0 to call inverse:

```
inverse(a0);
```

The system only needs to pass the address of *a0.data* as a static array. This becomes:

```
inverse(a0.data);
```

Consider another example:

```
void inverse2(double a[m][n] /static, int m, int n);
```

Function *inverse* takes a static arra *a* and parameters *m* and *n*, which denote the number of rows and the number of columns, respectively. We define another function *inverse* , which takes a dynamic arra *a*. Function *inverse* can be implemented by a call to *inverse* :

```
void inverse3(double a[][] /dynamic)
{
    inverse2(a, a.length[0], a.length[1]);
}
```

a.length[0] and *a.length*[1] denote the number of rows and the number of columns of array *a*. Note that the base of the arra *a.length* is 0.

4. SUBSCRIPT ASSIGNMENT

Python's subscript operators are classified into three categories: *get*, *assi* , and *delete*. The *Dictionary* data type is a set of pairs of keys and values, which data types are denoted as *Key* and *Value*, respectively. Let *d* be an object of *Dictionary*, *Key* be *String*, and *Value* be *Object*. The following are examples of using *Dictionary*:

```
Ex. 1:    if (d["machine"] == "Apollo") ... ;
Ex. 2:    d["cpu load"] = 0.18;
Ex. 3:    delete d["erroneous jobs"];
Ex. 4:    d["erroneous jobs"] = null;
```

Examples 1 and 2 use subscript *get* and subscript *assi* . Example 3 uses subscript *delete*. Example 4 is a special case of the subscript assign, where the assigned value is *null*. In runtime environments with automatic garbage collection, Examples 3 and 4 have the same semantics; however, Python allows they to be defined as different.

To preserve the original semantics of the delete operator and to avoid introducing the ambiguities in Examples 3 and 4, we do not allow the delete operator. An array interface consists of only two kinds of subscript operators: *get* and *assi* . The class *Dictionary* using the syntax of C++ is as follows:

```
class Dictionary
{
  public: ...
     Value operator[] (Key k);
     void operator[]= (Key k, Value v);
};
```

The operator '[]' represents the subscript operato *get*, which is the original C++ subscript operator. Because operator '[]' can only be used in the right-hand side of an assignment, *Value* does not need to be a reference data type. The operator '[]=' represents subscript operato *assi* , which is new to C++. These operators

These subscript operators can also be overloaded. Consider a mutable list class *Sequence*:

```
class Sequence
{
  public: ...
     Value operator[] (Key k);
     Value[] operator[] (Slice<Key> s);
     void operator[]= (Key k, Value v);
     void operator[]= (Slice<Key> s, Value v);
};
```

There are two method names of operator '[]' and operator '[]='. "Slice<Key> s" denotes the index data type is a slice of Key. "Value[]" denotes an array of *Key*.

5. Multidimensional Index

Subscript operators can be extended to be multidimensional. Use Matrix as an example:

```
class Matrix
{
  public:
     Matrix (int dims[]);
     double operator[] (int index[]);
     void operator[]= (int index[], double v);
};
```

Here the *index* is an array. The following are examples of uses:

```
Matrix m[250, 250];    // dims = [250, 250];
if (m[i, i+1] > 0.0) ... ; // operator[], index = [i, i+1];
m[i, j] = 0.0;             // operator[]=, index = [i, j];
```

A more flexible approach is dividing the index into an equal number of parameters:

```
class Array2
{
  public:
     Array2 (int i, int j);
     double operator[] (int i, int j);
     void operator[]= (int i, int j, double v);
};
```

Array2 is a 2-dimensional array. A 2-dimensional index is divided into two parameters: *i* and *j*. Consider the following examples of use:

```
Array2 a[250, 250];   // (i, j) = (250, 250);
if (a[1, 2] > 0.0) ... ;  // operator[], (i, j) =(1, 2);
a[2, 3] = 0.0;            // operator[]=, (i, j, v) = (2, 3, 0.0);
```

This approach is useful for arrays with a fixed number of dimensions, where the index of each dimension can be directly given with variables such as *i*, *j*, and *k*.

Both approaches introduce no syntax extension but solve the fundamental problem of subscript operators in C++ and Java. This feature meets one of the design goals of Java: simplicity and security. Firstly, for C++ and Java, this feature uses only two operators [] and []=, which are the least extension known. Secondly, this feature does not adopt reference types, so there is no danger for destruction of object's internal data structures. Thirdly, a list of indexes separated by commas in subscript operators is considered to be a multidimensional index rathe than a confusing comma expression in C and C++. In addition, syntactically operator '[]=' is similar to other C and C++ assignment operators, such as '+=', '*=', etc. This follows the convention of C++'s naming style. P ogrammers familiar to C++ would be familiar to this extension as well.

6. Conclusions and Future Work

This paper has presented the following language features: 1) Layout annotation: An array declaration can be annotated with binding and la yout order of the array. 2) Subscript assignment: An object interface consists of two subscript operators: *get* and *assi* . 3) Multidimensional index: We extend subscript operators to be multidimensional. These language features make array interfaces safe and powerful. The features presented also indicate that recent languages such as C++ and Java still have weak points, which can be much improved with acceptable effort. In the future, research effort should also emphasize on integrating existing programming features and best programming practices.

Reference

[1] Adams, Jeanne C. et al., *FORTRAN 90 Handbook - Complete ANSI/ISO Reference*, New York, Intertext Publications, 1992.

[2] Booch, G., *Software Engineering with Ada* , The Benjamin/Cummings Publishing Company, Inc., California, 1983.

[3] Ellis, M.A., Stroustrup, B., *The Annotated C++ Reference Manual*, Addison-Wesley, Massachusetts, 1990.

[4] Flanagan, D., *Java in a Nutshell*, O-Reilly & Associates, Inc., 1996.

[5] Ghezzi, C., Jazayeri, M., *Programming Language Concepts*, 2nd Ed., John Wiley & Sons, Inc., 1987.

[6] Goldberg, A., Robson, D., *Smalltalk-80: The Language*, Addison-Wesley, Massachusetts, 1989.

[7] Grogono, P., *Programming in Pascal*, 2nd Ed., 1984.

[8] Kernighan, B.W., and Ritchie, D. M., *The C Programming Language*, 2nd Ed., Prentice-Hall, New Jersey, 1988.

[9] Lutz, M., *Programming Python*, O'Reilly & Associates, Inc., 1996.

[10] Meyer, B., *Object-oriented Software Construction*, Prentice-Hall, New York, 1988.