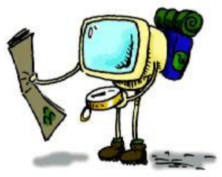
Programming Languages

Tevfik Koşar

Lecture - XVII
March 21st , 2006

Roadmap

- Variant Records (Unions)
- Arrays



2

Variant Records (Unions)

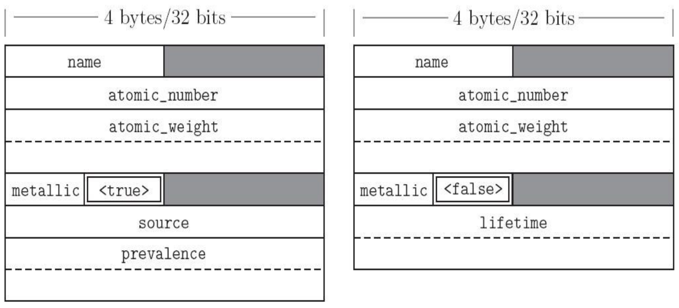


Figure 7.4: **Likely memory layouts for `element` variants.** The value of the `naturally_occurring` field (shown here with a double border) determines which of the interpretations of the remaining space is valid. Type `string_ptr` is assumed to be represented by a (four-byte) pointer to dynamically allocated storage.

3

Variant Records (Unions)

```
type tag = (is_int, is_real, is_bool);
var test: record
    case which: tag of
            is_int     : (i: integer);
            is_real: (r:real);
            is_bool: (b:Boolean);
end;
----

test.which : = is_real;
test.r := 3.0;
test.which:= is_int;
writeln(test.i);
```

Not an error, but the output will be junk!

4

Variant Records (Unions)

```
type tag = (is_int, is_real, is_bool);
var test: record
    case tag of
            is_int     : (i: integer);
            is_real: (r:real);
            is_bool: (b:Boolean);
end;
----
```

X test.which : = is_real; → not required

```
test.r := 3.0;

writeln(test.i);
```

Not an error, but the output will be junk!

5

Variant Records (Unions)

- Variant records with tags: discriminated unons
- Variant records without tags: nondiscriminated unions

6

Variant Records (Unions)

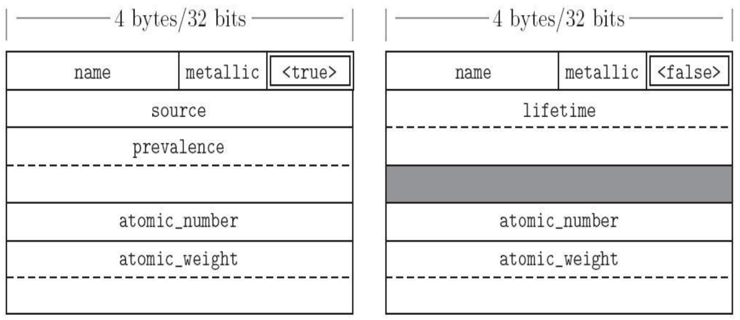


Figure 7.5: **Likely memory layout for a variant record in Modula-2.** Here the variant portion of the record is not required to lie at the end. Every field has a fixed offset from the beginning of the record, with internal holes as necessary following small-size variants.

7

Arrays

- Arrays are the most common and important composite data types
- Unlike records, which group related fields of disparate types, arrays are usually homogeneous
- Semantically, they can be thought of as a mapping from an *index type* to a *component* or *element type*
- A *slice* or *section* is a rectangular portion of an array

8

Arrays

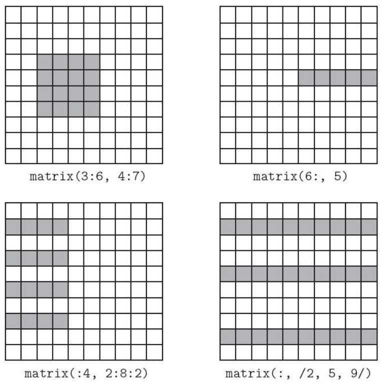


Figure 7.6: **Array slices (sections) in Fortran 90.** Much like the values in the header of an enumeration-controlled loop (Section 6.5.1), $a : b : c$ in a subscript indicates positions a, $a + c$, $a + 2c$, ... through b. If a or b is omitted, the corresponding bound of the array is assumed. If c is omitted, `1` is assumed. It is even possible to use negative values of c in order to select positions in reverse order. The slashes in the second subscript of the lower right example delimit an explicit list of positions.

Arrays

- Dimensions, Bounds, and Allocation
  - ***global lifetime, static shape*** — If the shape of an array is known at compile time, and if the array can exist throughout the execution of the program, then the compiler can allocate space for the array in static global memory
  - ***local lifetime, static shape*** — If the shape of the array is known at compile time, but the array should not exist throughout the execution of the program, then space can be allocated in the subroutine’s stack frame at run time.
  - ***local lifetime, shape bound at elaboration time***

Arrays

```
procedure foo (size : integer) is
M : array (1..size, 1..size) of real;
...
begin
...
end foo;
```

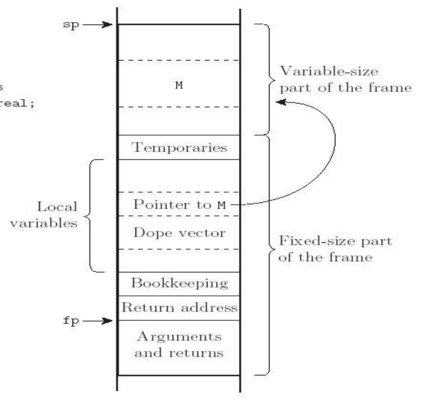


Figure 7.7: **Allocation in Ada of local arrays whose shape is bound at elaboration time.** Here `M` is a square two-dimensional array whose width is determined by a parameter passed to `foo` at run time. The compiler arranges for a pointer to `M` to reside at a static offset from the frame pointer. `M` cannot be placed among the other local variables because it would prevent those higher in the frame from having static offsets. Additional variable-size arrays are easily accommodated. The purpose of the *dope vector* field is explained in Section 7.4.3.

Arrays

- Contiguous elements
  - column major - only in Fortran
  - row major
    - used by everybody else
    - makes array [a..b, c..d] the same as array [a..b] of array [c..d]

12

Arrays

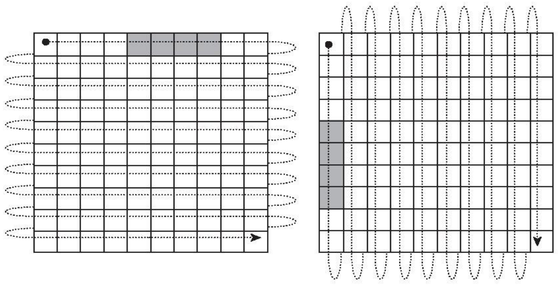


Figure 7.9: **Row- and column-major memory layout for two-dimensional arrays.** In row-major order, the elements of a row are contiguous in memory; in column-major order, the elements of a column are contiguous. The second cache line of each array is shaded, on the assumption that each element is an eight-byte floating-point number, that cache lines are 32 bytes long (a common size), and that the array begins at a cache line boundary. If the array is indexed from A[0,0] to A[9,0], then in the row-major case elements A[0,4] through A[0,7] share a cache line; in the column-major case elements A[4,0] through A[7,0] share a cache line.