Chapter 7

Data types

February 23, Lecture 11

Arrays

- One of the most important data type construct
- Referring to one element
  - A(3) : Ada
  - A[3]: Pascal and C

- Declarations

```
character, dimension (1:26) :: upper
character (26) upper    ! shorthand notation
```

```
char upper[26];
```

```
var upper : array ['a'..'z'] of char;
```

```
upper : array (character range 'a'..'z') of character;
```

Arrays

- Multidimensional arrays

```
matrix : array (1..10, 1..10) of real;     -- Ada

real, dimension (10,10) :: matrix          ! Fortran
```

- In Modula-3

```
VAR matrix : ARRAY [1..10], [1..10] OF REAL;
```

is syntactic sugar for

```
VAR matrix : ARRAY [1..10] OF ARRAY [1..10] OF REAL;
```

Arrays

- In Ada

In Ada, by contrast,

```
matrix : array (1..10, 1..10) of real;
```

is not the same as

```
matrix : array (1..10) of array (1..10) of real;
```

- matrix(3,4)
- matrix(3)(4)

Arrays

- In Ada

In Ada, by contrast,

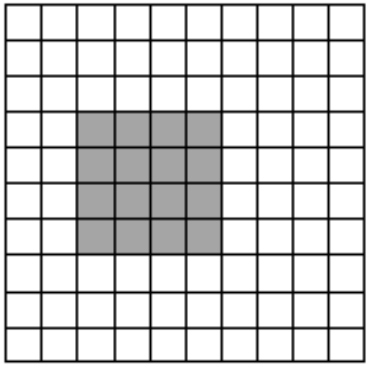
```
matrix : array (1..10, 1..10) of real;
```

is not the same as

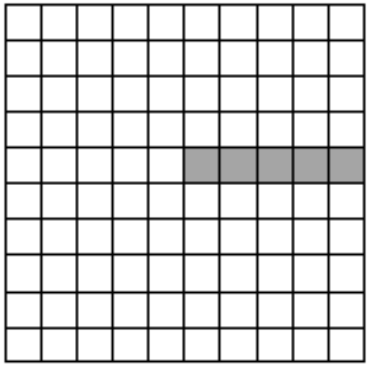
```
matrix : array (1..10) of array (1..10) of real;
```

- The first definition supports nice things like
  - Accessing submatrices

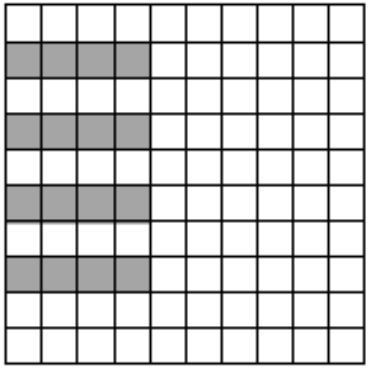
Arrays: slicing



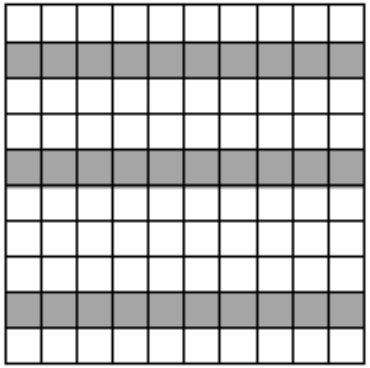
matrix(3:6, 4:7)



matrix(6:, 5)



matrix(:4, 2:8:2)



matrix(:, (/2, 5, 9/))

Arrays: operators

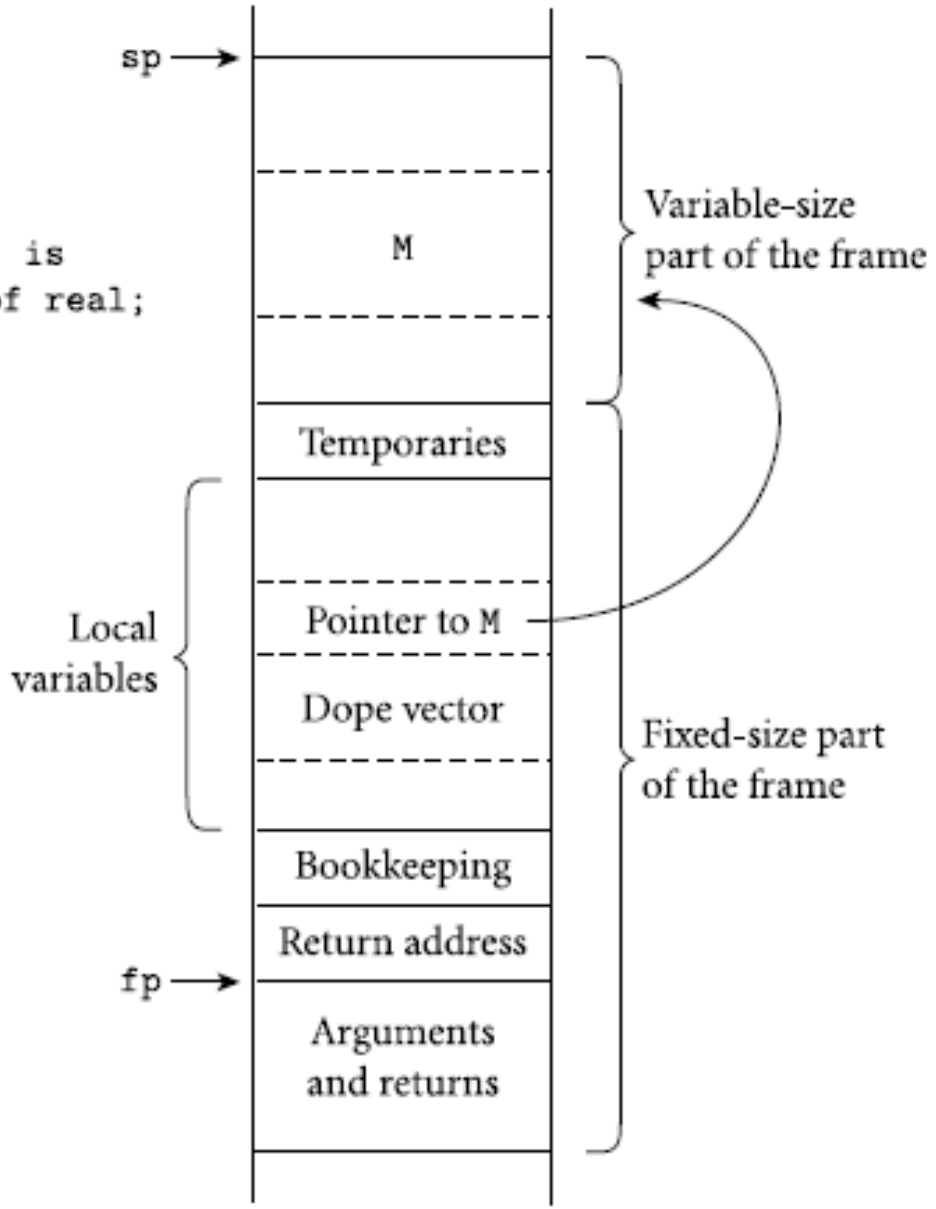
- Some languages support equality
- Some languages (notably Fortran 90, matlab) support many operations. Arrays should be of the same shape.
- Slices of the same shape can be intermixed.
- Intrinsic functions defined on arrays

Arrays: Dimensions, bounds, allocation

- We saw declarations of dimensions at compile time
- But the dimensions may be not known at compile

- There are five cases:
  1. Global lifetime, static shape: static global memory
  2. Local lifetime, static shape: subroutine's stack frame
  3. Local lifetime, shape bound at elaboration time: extra level of indirection
     - Stack is divided into fixed-size part and variable-size part

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



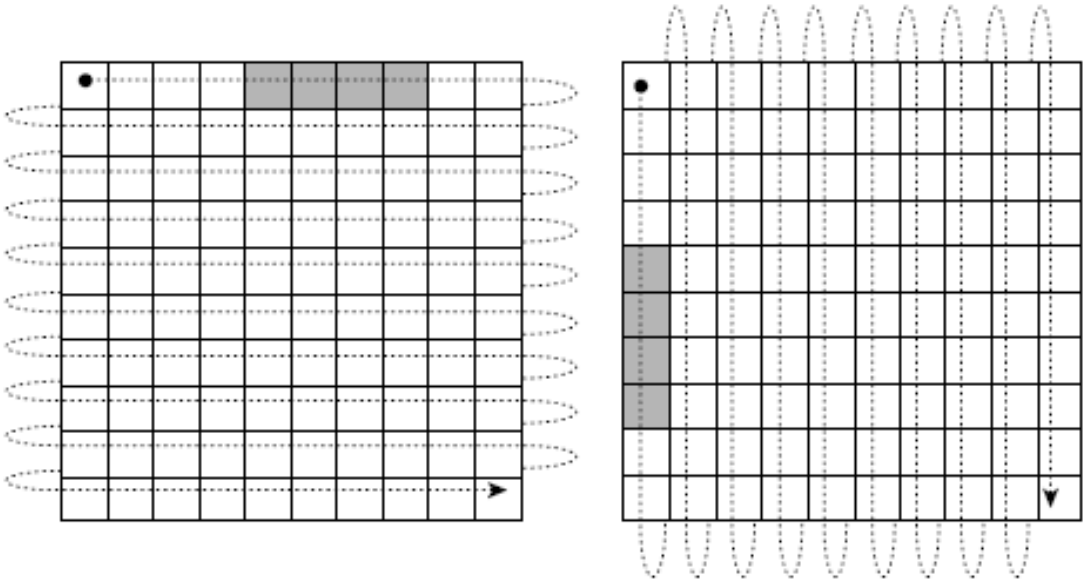
Arrays: Dimensions, bounds, allocation

- We saw declarations of dimensions at compile time
- But the dimensions may be not known at compile

- There are five cases:
  1. Global lifetime, static shape: static global memory
  2. Local lifetime, static shape: subroutine's stack frame
  3. Local lifetime, shape bound at elaboration time: extra level of indirection
     - Stack is divided into fixed-size part and variable-size part
  4. Arbitrary lifetime, shape bound at elaboration time: heap
     - This is what happens in Java, C#. Evey array is a reference to an object, in the object oriented sense.
  5. Arbitrary lifetime, dynamic shape: heap again, pointer to it from the frame. When more space is needed: allocate new, copy, de-allocate old

Arrays: Memory layout

- Row-major vs column-major order
- Significance for cache misses



Row-major order

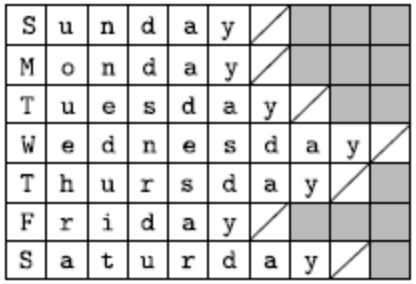
Column-major order

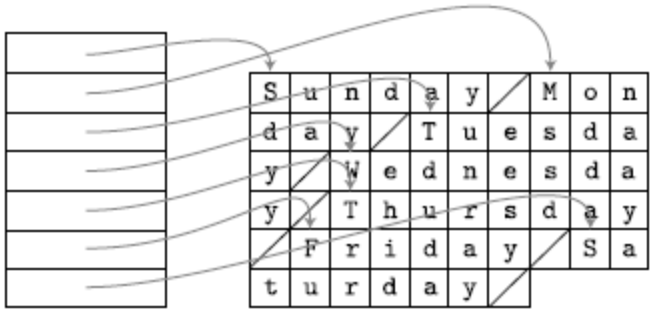
Arrays: Memory layout

- Row-Pointer Layout
- It allows individual access, may save space
- Use preexisting rows

```
char days[][10] = {
    "Sunday", "Monday", "Tuesday",
    "Wednesday", "Thursday",
    "Friday", "Saturday"
};
...
days[2][3] == 's';  /* in Tuesday */
```

```
char *days[] = {
    "Sunday", "Monday", "Tuesday",
    "Wednesday", "Thursday",
    "Friday", "Saturday"
};
...
days[2][3] == 's';  /* in Tuesday */
```





Arrays: Other topics

- Calculating addresses
- Sparse arrays