

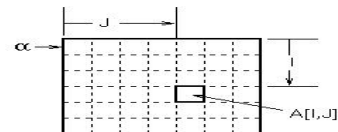
Array accessing

- An array is an ordered sequence of identical objects.
- The ordering is determined by a scalar data object (usually integer or enumeration data). This value is called the **subscript** or **index**, and written as $A[I]$ for array A and subscript I .
- **Multidimensional arrays have more than one subscript.** A 2-dimensional array can be modeled as the boxes on a rectangular grid.
- The L-value for array element $A[I,J]$ is given by the accessing formula on the next slide

CMSC430 Spring 2007

1

Array storage



$$L\text{-value}(A[I,J]) = \alpha + \text{skip } I \text{ rows} + j \text{ columns}$$

Actual storage: $A[L1:U1, L2:U2]$



$$L\text{-value}(A[I,J]) = \alpha + \text{number of rows} * \text{rowsize} + \text{number of columns} * \text{elementsize}$$

$$= \alpha + a * d1 + b * d2$$

$d2$ = element size
 4 – usually for integer
 4 – usually for float
 1 – usually for char

$$d1 = \text{NumberElements} * \text{elementsize} - (U2 - L2 + 1) * d2$$

Rows to skip: $(I - L1)$
 Columns to skip: $(J - L2)$

$$L\text{-value}(A[I,J]) = \alpha + d1 * (I - L1) + d2 * (J - L2)$$

CMSC430 Spring 2007

2

Array accessing (continued)

Rewriting access equation:

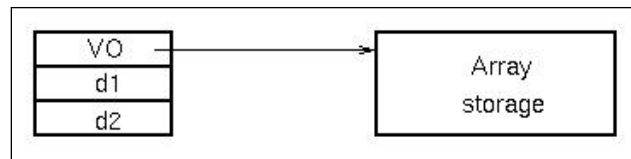
$$L\text{-value}(A[I,J]) = \alpha - d1*L1 - d2*L2 + I*d1 + J*d2$$

Set $I = 0$; $J = 0$;

$$L\text{-value}(A[0,0]) = \alpha - d1*L1 - d2*L2 + 0*d1 + 0*d2$$

$L\text{-value}(A[0,0]) = \alpha - d1*L1 - d2*L2$, which is a constant.

- Call this constant the **virtual origin (VO)**; It represents the address of the 0th element of the array.
 $L\text{-value}(A[I,J]) = VO + I*d1 + J*d2$
- To access an array element, typically use a **dope vector**:



CMSC430 Spring 2007

3

Array accessing summary

To create arrays:

1. Allocate total storage beginning at α :

$$(U2-L2+1)*(U1-L1+1)*\text{elsize}$$

2. $d2 = \text{elsize}$

3. $d1 = (U2-L2+1)*d2$

4. $VO = \alpha - L1*d1 - L2*d2$

5. To access $A[I,J]$: $L\text{value}(A[I,J]) = VO + I*d1 + J*d2$

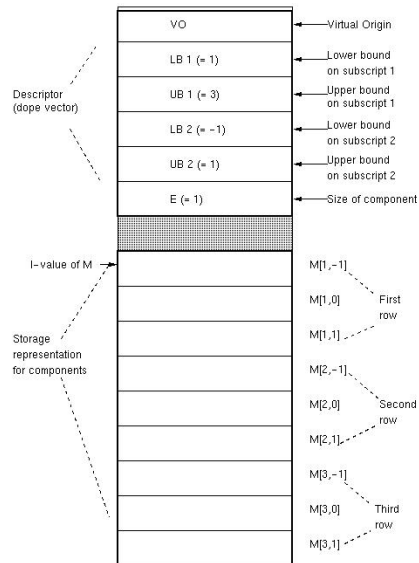
This works for 1, 2 or more dimensions.

- May not require runtime dope vector if all values are known at compile time. (e.g., in Pascal, $d1$, $d2$, and VO can be computed by compiler.)
- Next slide: Storage representation for 2-dimensional array.

CMSC430 Spring 2007

4

Array storage representation



5

Array example

Given following array: var A: array [7..12, 14..16] of real;

- Give dope vector if array stored beginning at location 500.

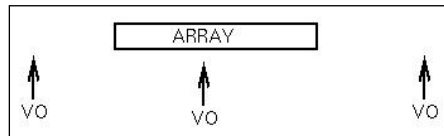
$d2 = 4$ (real data)

$d1 = (16-14+1) * 4 = 3 * 4 = 12$

$VO = 500 - 7 * 12 - 14 * 4 = 500 - 84 - 56 = 360$

$L\text{-value}(A[I,J]) = 360 + 12 * I + 4 * J$

1. VO can be a positive or negative value, and can have an address that is before, within, or after the actual storage for the array:

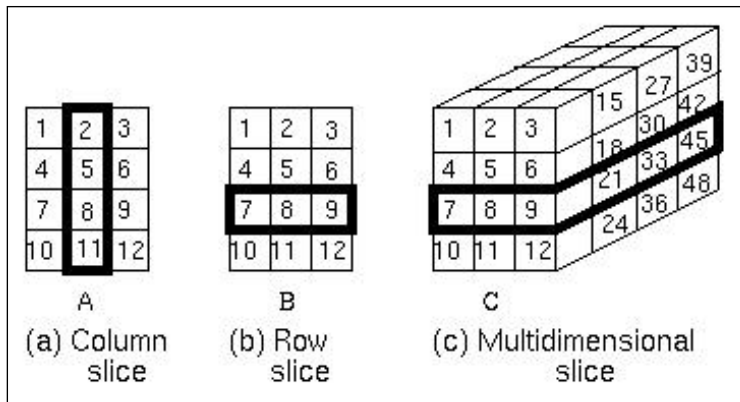


2. In C, $VO = \alpha$ since bounds start at 0.

Example: char A[25]

$L\text{-value}(A[I]) = VO + (I-1) * d1 = \alpha + I * 1 = \alpha + I$

Slices

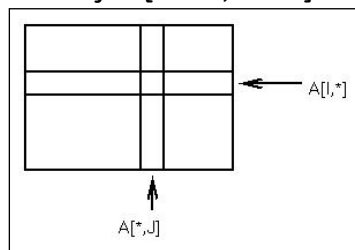


CMSC430 Spring 2007

7

Array slices

Given array : $A[L1:U1, L2:U2]$: Give $d1$, $d2$, and VO for vector:



Dope vector $A[I,*] = B[L2:U2]$
 $VO = L\text{-value}(A[I,L2]) - d2 * L2$
 $M1 = \text{elsize} = d2$

Create new dope vector
that accesses original
data

Dope vector $A[* ,J] = B[L1:U1]$
 $VO = L\text{-value}(A[L1,J]) - d1 * L1$
 $M1 = \text{rowsize} = d1$

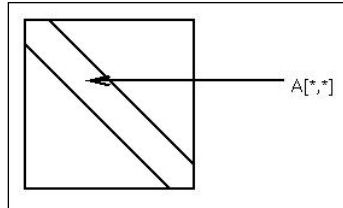
CMSC430 Spring 2007

8

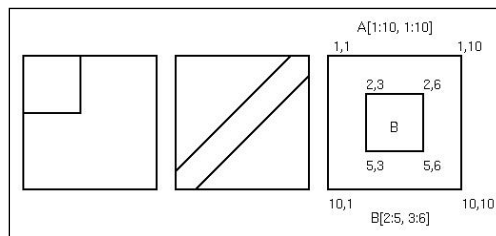
More on slices

- **Diagonal slices:**

$$\begin{aligned} VO &= L\text{-value}(A[L1,L2]) \\ &\quad - d1*L1 - d2*L2 \\ M1 &= d1 + d2 \end{aligned}$$



- **Other possibilities:**



CMSC430 Spring 2007

9

Associative arrays

Access information by name without having a predefined ordering or enumeration:

- Example: Names and grades for students in a class:
NAME[I] = name of Ith student
GRADE[I] = Grade for Ith student

Associative array: Use Name as index:
CLASS[name] will be grade.

Problem: Do not know enumeration before obtaining data so dope vector method of accessing will not work.

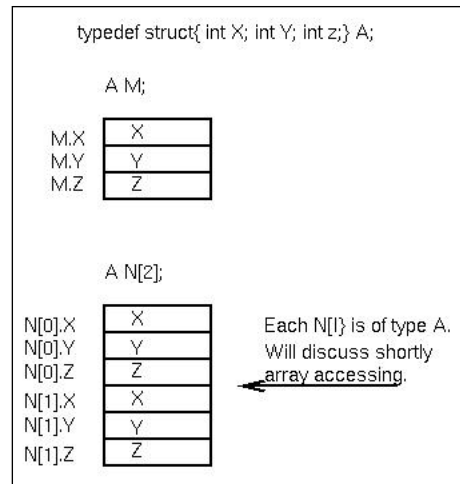
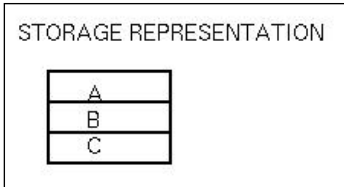
Implemented in Perl and in SNOBOL4 (as a table)

CMSC430 Spring 2007

10

Structs in C

- **Representation:** a sequence of objects:
 record { A: object;
 B: object;
 C: object }



CMSC430 Spring 2007

11

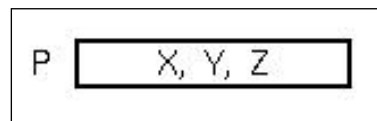
Union types

```
typedef union { int X;  

  float Y;  

  char Z[4];} B;
```

B P;



- Similar to records, except all have overlapping (same) L-value.
- But problems can occur. What happens below?
 P.X = 142;
 printf("%O\n", P.Z[3])
 All 3 data objects have same L-value and occupy same storage. No enforcement of type checking.
 ⇒ Poor language design

CMSC430 Spring 2007

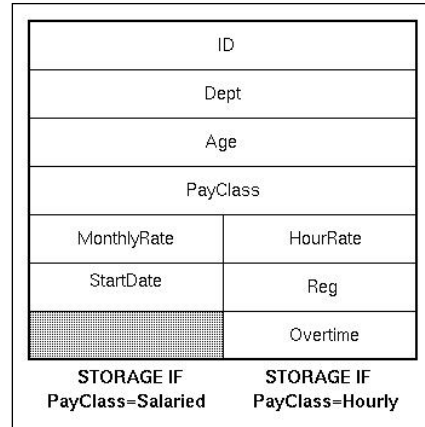
12

Variant records

```

type PayType=(Salaried, Hourly);
var Employee:record
  ID: integer;
  Dept: array[1..3] of char;
  Age: integer;
  case PayClass: PayType of
    Salaried:(MonthlyRate:real;
              StartDate:integer);
    Hourly:(HourRate:real;
            Reg:integer;
            Overtime:integer)
  end

```



CMSC430 Spring 2007

13

Variant records (continued)

Tagged union type - Pascal variant records

```

type whichtype = (inttype, realltype, chartype);
type uniontype = record
  case V: whichtype of
    inttype: (X: integer);
    realltype: (Y: real);
    chartype: (Z: char4); Assumes string of length 4
  end

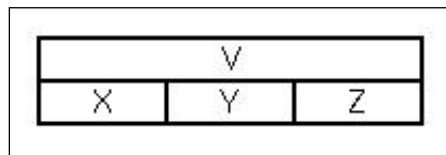
```

But can still subvert tagging:

```

var P: uniontype
P.V = inttype;
P.X = 142;
P.V = chartype;

```



CMSC430 Spring 2007
What is P.V value now?

14