CSE 142
Programming I

Arrays

Chapter 8

Motivation: Sorting

Input: 10 15 4 25 17 3 12 36 48 32 9 21

Desired output:
3 4 9 10 12 15 17 21 25 32 36 48

How can this be done?

If we had lots of variables we could store each input in a variable.

But think about what the program would be like.

Is there a better way?

Another Motivation - Averaging Grades

```
double grade1, grade2, grade3, grade4, grade5,
        grade6, grade7, total ;

/* initialize grades somehow...*/

total = grade1 + grade2 + grade3 + grade4
        + grade5 + grade6 + grade7 ;

printf( “average =  %f \n”, total /  7.0) ;
```

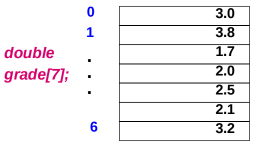
What if we had 500 grades to add up instead of 7?

Data Structures

- **Functions give us a way to organize programs.**
- **Data structures are needed to organize data, especially:**
  - **large amounts of data**
  - **variable amounts of data**
  - **sets of data where the individual pieces are related to one another**
- **In this course, we will structure data using**
  - **arrays**
  - ***structs***
  - **combinations of arrays and *structs***

Arrays

- **Definition: *A named, ordered collection of values of identical type***
- **Name the collection *(grade)*; number the elements *(0 to 6)***
- **Example: grades for 7 students**



C expressions:

***grade[0]* is 3.0**

***grade[6]* is 3.2**

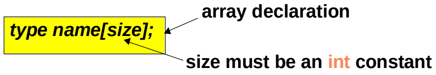
***2.0*grade[3]* is 4.0**

...

Averaging Grades II

```
#define MAXGRADES 7
double grade[MAXGRADES], total ;
int index;
... /* initialize grades somehow... then:
total = grade[0] + grade[1] + grade[2] + grade[3]
         + grade[4] + grade[5] + grade[6];
   or here's how we really would do it: */
total = 0;
for( index=0; index<MAXGRADES; index++) {
      total = total + grade[index];
}
printf( "average =  %f \n", total /  MAXGRADES) ;
```

Array Terminology



type name[size];

array declaration

size must be an int constant

double grade[7];

- *grade* is of type array of double with size 7.
- *grade[0], grade[1], ... , grade[6]* are the elements of the array *grade*. Each is of type double.
- *0,1, ... , 6* are the indices of the array. Also called subscripts.
- The bounds are the lowest and highest values of the subscripts (here: 0 and 6).

Array names are identifiers

- **Therefore:**
  - **They follow the all usual rules for C identifiers (start with a letter, etc.)**
  - **They must be declared before they are used**
- **If you see *x[y]* in a program, then you know that**
  - ***x* should be the name of an array**
  - ***y* should have an integer value**

Index Rule

Rule: *An array index must evaluate to an int between 0 and n-1, where n is the number of elements in the array.* No exceptions!

Example:

```
    grade[i+3+k]        /* OK as long as 0 ≤ i+3+k ≤ 6 */
```

The index may be very simple

```
    grade[0]
```

or incredibly complex

```
    grade[(int) (3.1 * fabs(sin (2.0*PI*sqrt(29.067))))]
```

C Array Bounds are Not Checked

```
#define CLASS_SIZE 7

double grade[CLASS_SIZE] ;
int index ;
index = 9 ;
...
grade[index ] = 3.5 ;          /* Is i out of range?? */
```

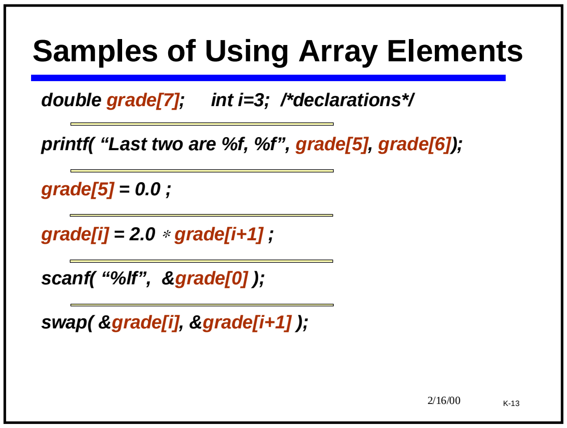
```
if ( 0 <= index  && index < CLASS_SIZE ) {
    grade[index ] = 3.5 ;
} else {
    printf("Array index %d out of range. \n", index ) ;
}
```

Element Rule

Rule: *An array element can be used wherever a simple variable of the same type can be used.* No exceptions!

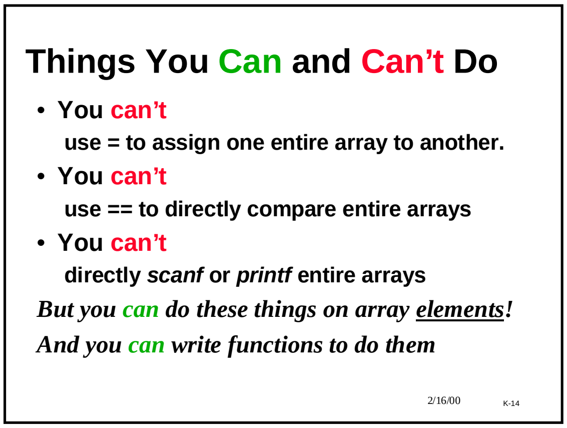
- **Examples:**

```
    scanf ( "%lf", &grade[i] ) ;

    grade[i] = sin (2.0 * PI * sqrt(29.067))
```



Samples of Using Array Elements

```
double grade[7];     int i=3;  /*declarations*/

printf( “Last two are %f, %f”, grade[5], grade[6]);

grade[5] = 0.0 ;

grade[i] = 2.0 * grade[i+1] ;

scanf( “%lf”,  &grade[0] );

swap( &grade[i], &grade[i+1] );
```

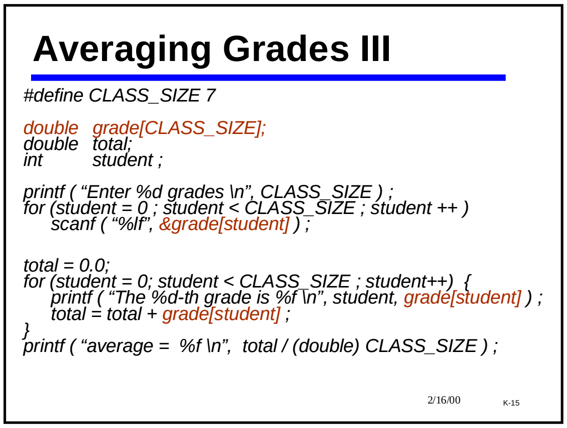


Things You Can and Can’t Do

- You can’t
  use = to assign one entire array to another.
- You can’t
  use == to directly compare entire arrays
- You can’t
  directly *scanf* or *printf* entire arrays

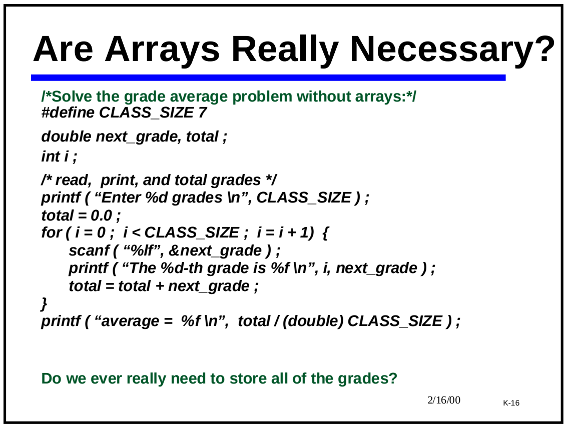
But you can do these things on array __elements__!

And you can write functions to do them



Averaging Grades III

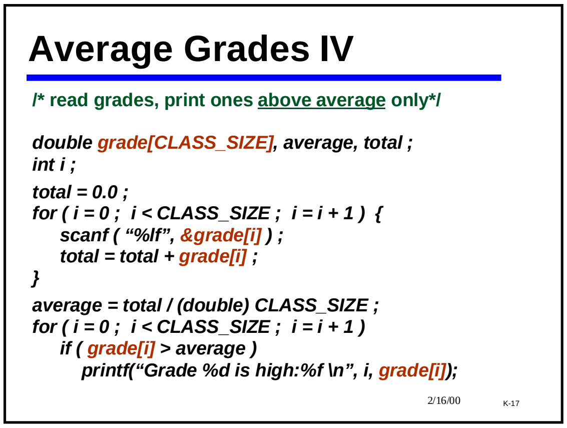
```
#define CLASS_SIZE 7

double   grade[CLASS_SIZE];
double   total;
int      student ;

printf ( “Enter %d grades \n”, CLASS_SIZE ) ;
for (student = 0 ; student < CLASS_SIZE ; student ++ )
     scanf ( “%lf”, &grade[student] ) ;


total = 0.0;
for (student = 0; student < CLASS_SIZE ; student++)  {
     printf ( “The %d-th grade is %f \n”, student, grade[student] ) ;
     total = total + grade[student] ;
}
printf ( “average =  %f \n”,  total / (double) CLASS_SIZE ) ;
```



Are Arrays Really Necessary?

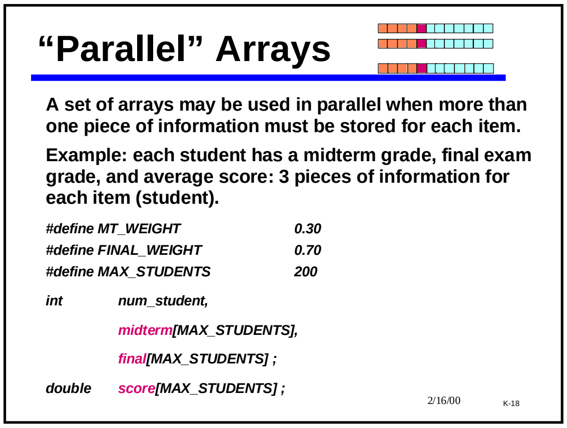
```
/*Solve the grade average problem without arrays:*/
#define CLASS_SIZE 7
double next_grade, total ;
int i ;
/* read,  print, and total grades */
printf ( “Enter %d grades \n”, CLASS_SIZE ) ;
total = 0.0 ;
for ( i = 0 ;  i < CLASS_SIZE ;  i = i + 1)  {
      scanf ( “%lf”, &next_grade ) ;
      printf ( “The %d-th grade is %f \n”, i, next_grade ) ;
      total = total + next_grade ;
}
printf ( “average =  %f \n”,  total / (double) CLASS_SIZE ) ;
```

Do we ever really need to store all of the grades?



Average Grades IV

```
/* read grades, print ones above average only*/

double grade[CLASS_SIZE], average, total ;
int i ;
total = 0.0 ;
for ( i = 0 ;  i < CLASS_SIZE ;  i = i + 1 )  {
     scanf ( “%lf”, &grade[i] ) ;
     total = total + grade[i] ;
}
average = total / (double) CLASS_SIZE ;
for ( i = 0 ;  i < CLASS_SIZE ;  i = i + 1 )
     if ( grade[i] > average )
          printf(“Grade %d is high:%f \n”, i, grade[i]);
```



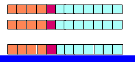
“Parallel” Arrays

A set of arrays may be used in parallel when more than one piece of information must be stored for each item.

Example: each student has a midterm grade, final exam grade, and average score: 3 pieces of information for each item (student).

```
#define MT_WEIGHT              0.30
#define FINAL_WEIGHT           0.70
#define MAX_STUDENTS           200

int        num_student,
           midterm[MAX_STUDENTS],
           final[MAX_STUDENTS] ;
double     score[MAX_STUDENTS] ;
```

Parallel Arrays



/* Suppose we have input the value of *num_students*, read student *i*'s grades for midterm and final, and stored them in *midterm[i]* and *final[i]*. Now:

Store a weighted average of exams in the array *score*.

*/

```
for ( i = 0 ;   i < num_student ;   i = i + 1 )  {
        score[i] = MT_WEIGHT * midterm[i] +
                  FINAL_WEIGHT * final[i] ;
}
```

Reading Array Elements

/* Read in student midterm and final grades and store them in two (parallel) arrays

*/

```
#define MAX_STUDENTS    200
int   midterm [MAX_STUDENTS] ;
int   final [MAX_STUDENTS] ;
int   num_student ;        /* actual number of students */
int   i, done, s_midterm, s_final ;
```

Reading Arrays

```
printf("Input number of students: ") ;
scanf("%d", &num_student) ;

if ( num_student > MAX_STUDENTS ) {
   printf("Too many students") ;
} else {
   for ( i = 0 ;  i < num_student ;  i = i+1 ) {
      scanf("%d %d", &midterm[i], &final[i]) ;
   }
}
```

Reading Arrays II

Terminate input with "sentinel" -1, -1

```
scanf("%d %d", &s_midterm, &s_final) ;

for {num_student = 0 ;
     s_midterm != -1 && num_student < MAX_STUDENTS;
     num_student++) {
        midterm[num_student] = s_midterm ;
        final[num_student] = s_final ;
        scanf("%d %d", &s_midterm, &s_final) ;
}
```

Keeping Track of the Elements In-Use

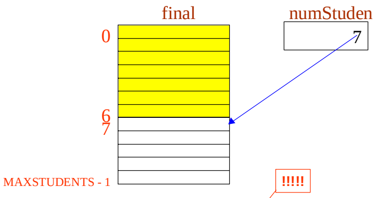
- Since the array has to be declared a fixed size, you often declare it bigger than you think you'll really need

```
#define MAXSTUDENTS 750
int      final[MAXSTUDENTS];
```

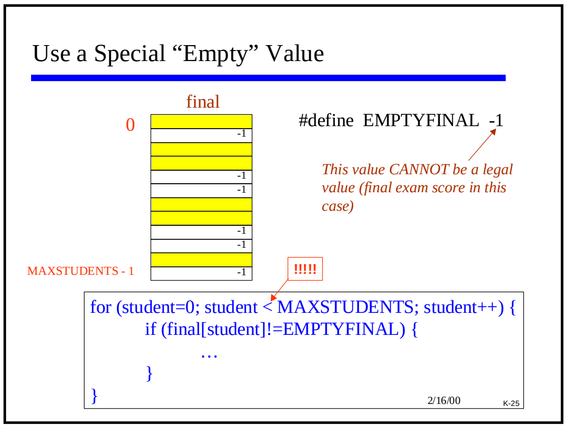
- How do you know which elements in the array actually hold data, and which are unused extras?

  1. Keep the valid entries together at the front
  2. Use a special value to denote "empty"
  3. Link the full entries together using parallel arrays

Keep the valid entries together



```
for (student=0; student < numStudents; student++) {
      …
}
```



Use a Special “Empty” Value

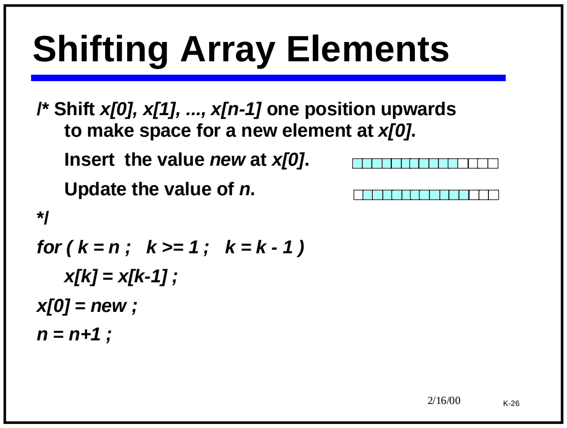
final
0
-1
-1
-1
-1
-1
MAXSTUDENTS - 1
-1
!!!!!

```
#define  EMPTYFINAL  -1
```

This value CANNOT be a legal value (final exam score in this case)

```
for (student=0; student < MAXSTUDENTS; student++) {
        if (final[student]!=EMPTYFINAL) {
                ...
        }
}
```

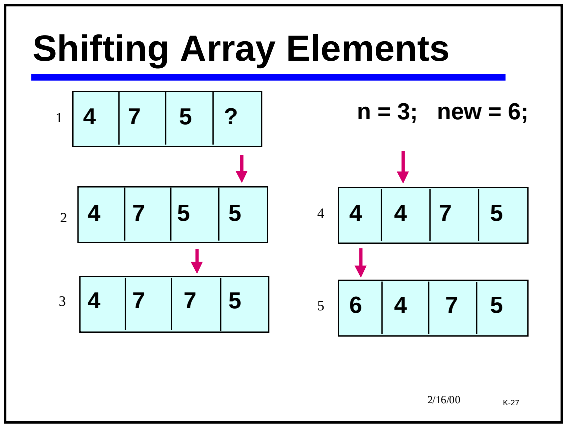
2/16/00 K-25



Shifting Array Elements

```
/* Shift x[0], x[1], ..., x[n-1] one position upwards
     to make space for a new element at x[0].
     Insert  the value new at x[0].
     Update the value of n.
*/
for ( k = n ;   k >= 1 ;   k = k - 1 )
     x[k] = x[k-1] ;
x[0] = new ;
n = n+1 ;
```

2/16/00 K-26



Shifting Array Elements

n = 3;   new = 6;

1 | 4 | 7 | 5 | ? |

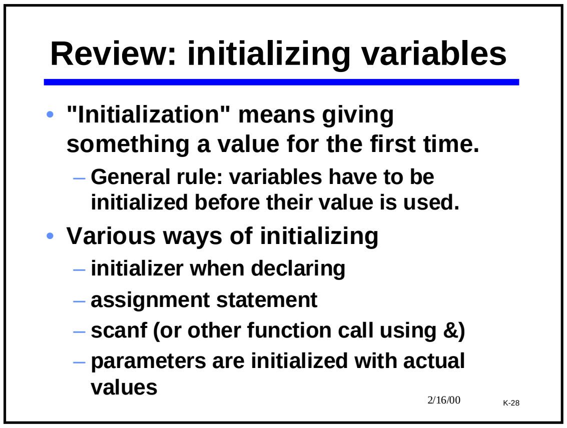
2 | 4 | 7 | 5 | 5 |

3 | 4 | 7 | 7 | 5 |

4 | 4 | 4 | 7 | 5 |

5 | 6 | 4 | 7 | 5 |

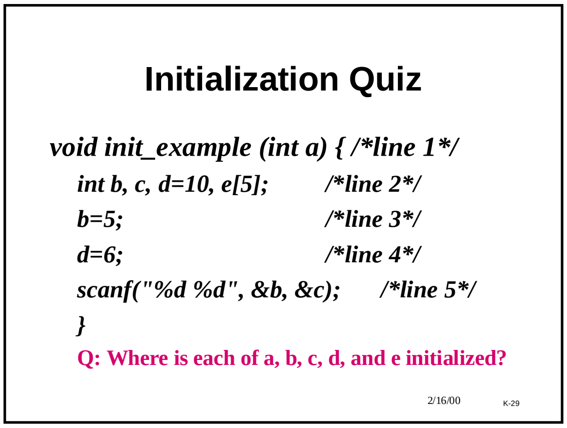
2/16/00 K-27



Review: initializing variables

- "Initialization" means giving something a value for the first time.
  - General rule: variables have to be initialized before their value is used.
- Various ways of initializing
  - initializer when declaring
  - assignment statement
  - scanf (or other function call using &)
  - parameters are initialized with actual values

2/16/00 K-28

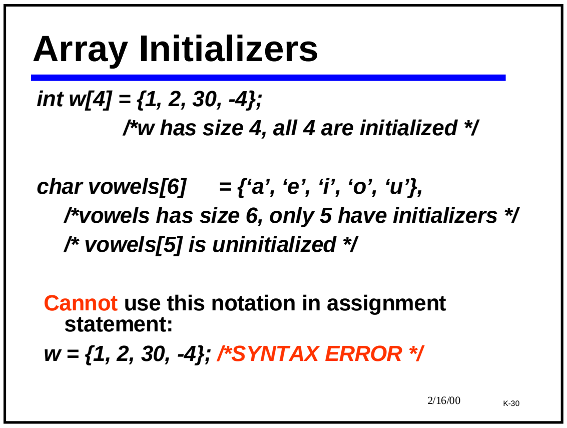


Initialization Quiz

```
void init_example (int a) { /*line 1*/
    int b, c, d=10, e[5];        /*line 2*/
    b=5;                         /*line 3*/
    d=6;                         /*line 4*/
    scanf("%d %d", &b, &c);      /*line 5*/
    }
```

Q: Where is each of a, b, c, d, and e initialized?

2/16/00 K-29



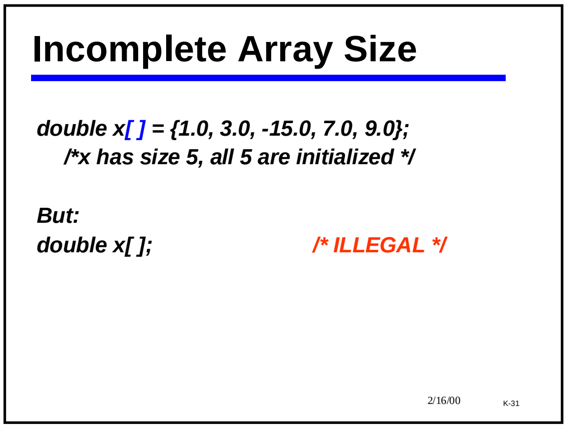
Array Initializers

```
int w[4] = {1, 2, 30, -4};
            /*w has size 4, all 4 are initialized */

char vowels[6]     = {‘a’, ‘e’, ‘i’, ‘o’, ‘u’},
    /*vowels has size 6, only 5 have initializers */
    /* vowels[5] is uninitialized */
```

Cannot use this notation in assignment statement:

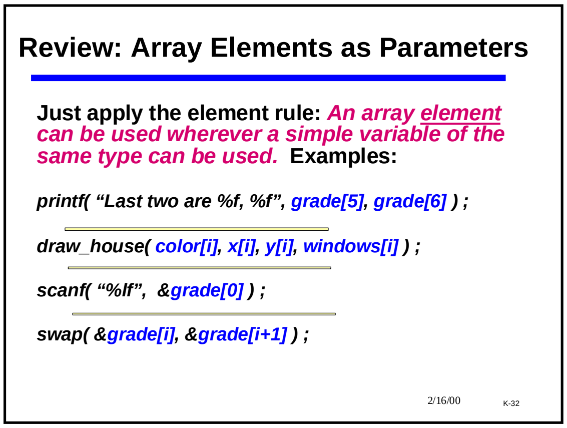
```
w = {1, 2, 30, -4}; /*SYNTAX ERROR */
```

2/16/00 K-30



Incomplete Array Size

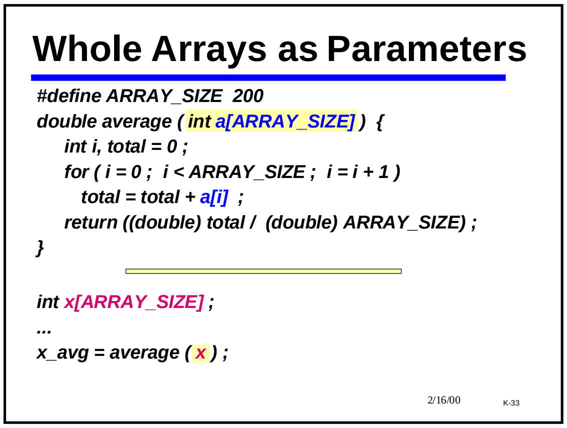
```
double x[ ] = {1.0, 3.0, -15.0, 7.0, 9.0};
   /*x has size 5, all 5 are initialized */

But:
double x[ ];                        /* ILLEGAL */
```



Review: Array Elements as Parameters

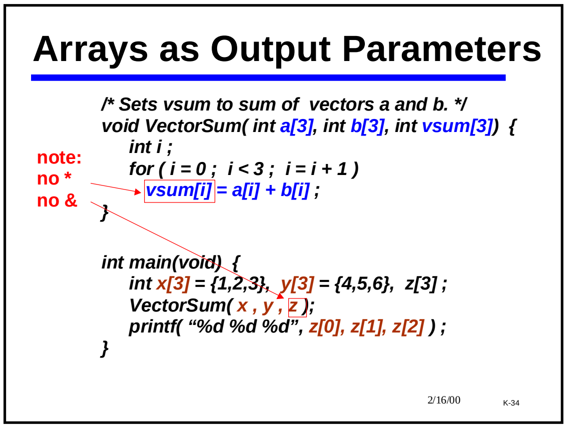
Just apply the element rule: *An array element can be used wherever a simple variable of the same type can be used.* Examples:

```
printf( "Last two are %f, %f", grade[5], grade[6] ) ;

draw_house( color[i], x[i], y[i], windows[i] ) ;

scanf( "%lf",  &grade[0] ) ;

swap( &grade[i], &grade[i+1] ) ;
```



Whole Arrays as Parameters

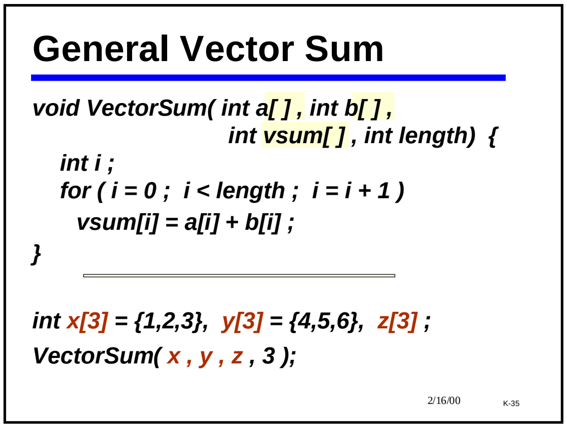
```
#define ARRAY_SIZE  200
double average ( int a[ARRAY_SIZE] )  {
     int i, total = 0 ;
     for ( i = 0 ;  i < ARRAY_SIZE ;  i = i + 1 )
        total = total + a[i]  ;
     return ((double) total /  (double) ARRAY_SIZE) ;
}

int x[ARRAY_SIZE] ;
...
x_avg = average ( x ) ;
```



Arrays as Output Parameters

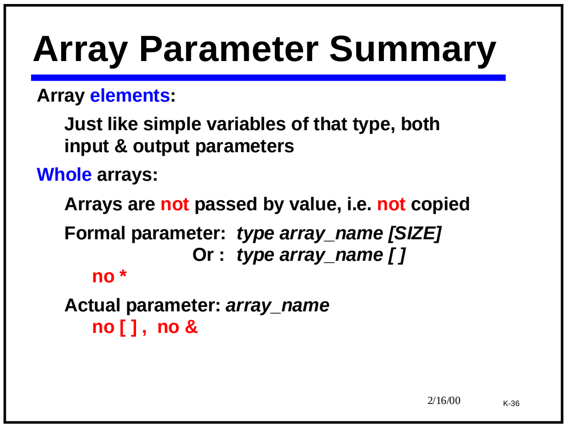
note:
no *
no &

```
/* Sets vsum to sum of  vectors a and b. */
void VectorSum( int a[3], int b[3], int vsum[3])  {
     int i ;
     for ( i = 0 ;  i < 3 ;  i = i + 1 )
        vsum[i] = a[i] + b[i] ;
}

int main(void)  {
     int x[3] = {1,2,3},  y[3] = {4,5,6},  z[3] ;
     VectorSum( x , y , z );
     printf( "%d %d %d", z[0], z[1], z[2] ) ;
}
```



General Vector Sum

```
void VectorSum( int a[ ] , int b[ ] ,
                      int vsum[ ] , int length)  {
   int i ;
   for ( i = 0 ;  i < length ;  i = i + 1 )
     vsum[i] = a[i] + b[i] ;
}

int x[3] = {1,2,3},  y[3] = {4,5,6},  z[3] ;
VectorSum( x , y , z , 3 );
```



Array Parameter Summary

Array elements:

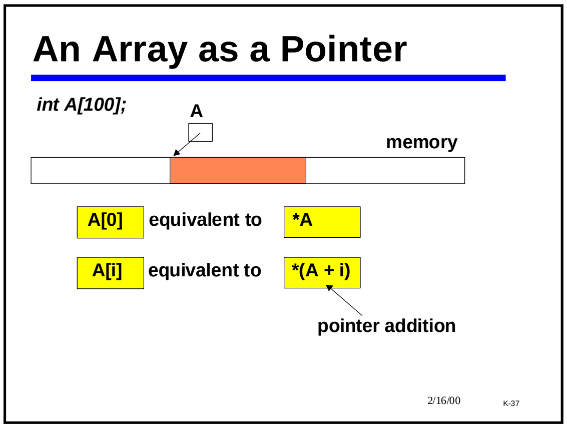
Just like simple variables of that type, both input & output parameters

Whole arrays:

Arrays are not passed by value, i.e. not copied

Formal parameter: *type array_name [SIZE]*
Or : *type array_name [ ]*
no *

Actual parameter: *array_name*
no [ ] , no &



An Array as a Pointer

int A[100];

A

memory

A[0] equivalent to *A

A[i] equivalent to *(A + i)

pointer addition