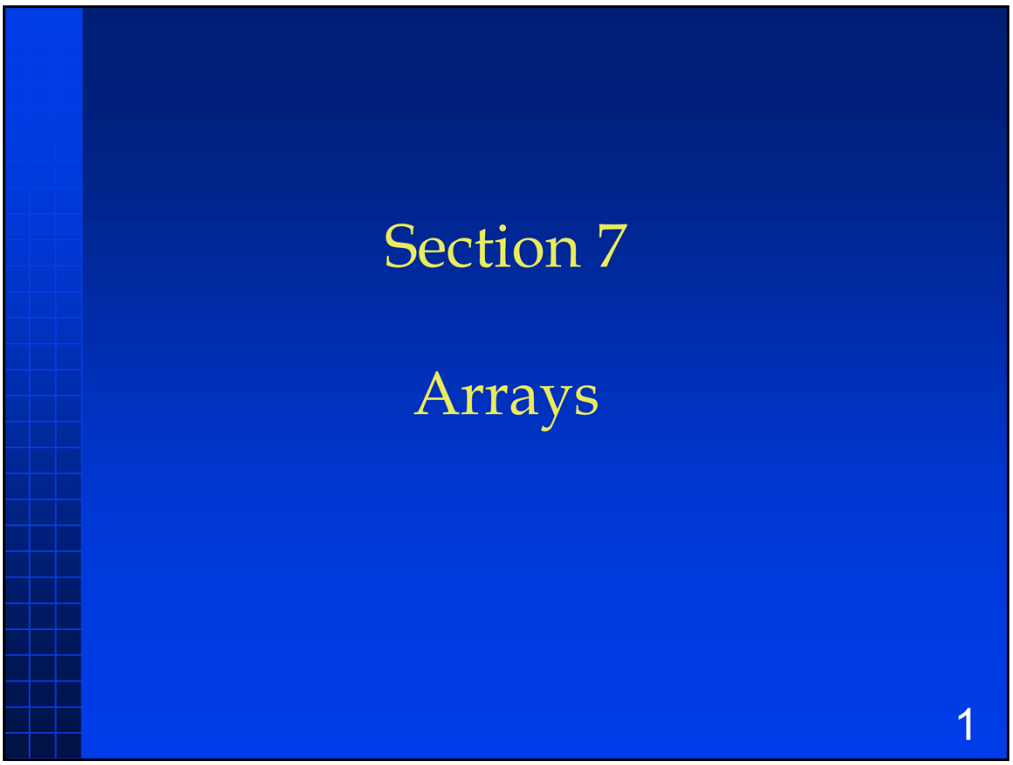



Arrays:

groups or collections of variables of the same type (homogeneous set), individual variables called elements. Elements are numbered: 0, 1, etc.

aka: vector, matrices, tensors

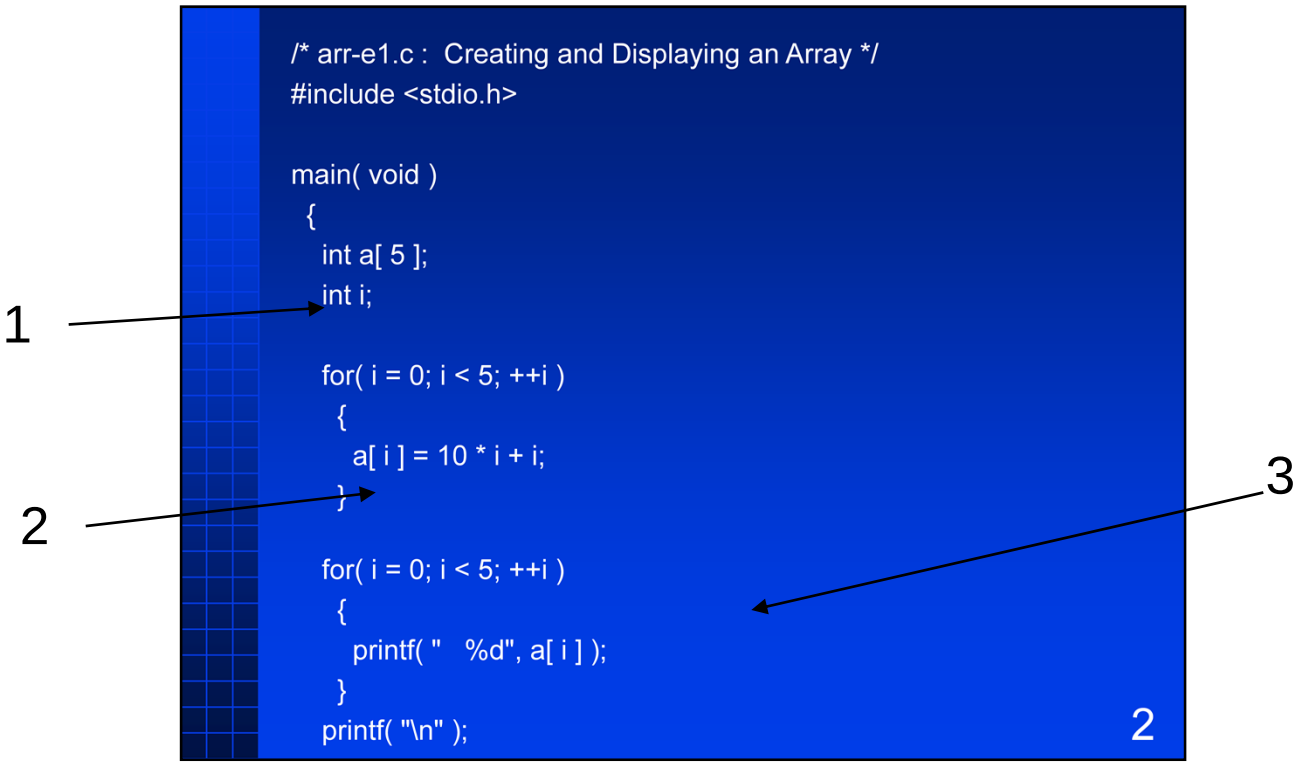
Have to be able to manipulate elements and entire arrays. Operation called subscripting which selects elements from entire arrays. Subscripting involves integer value that says which element selected

arrays are direct-access structures: can select elements in any order, no restrictions on access

In C, fixed # elements; not all elements have to be used (eg strings).

Numbering system for elements starts at 0, no choice.

No checking on bounds

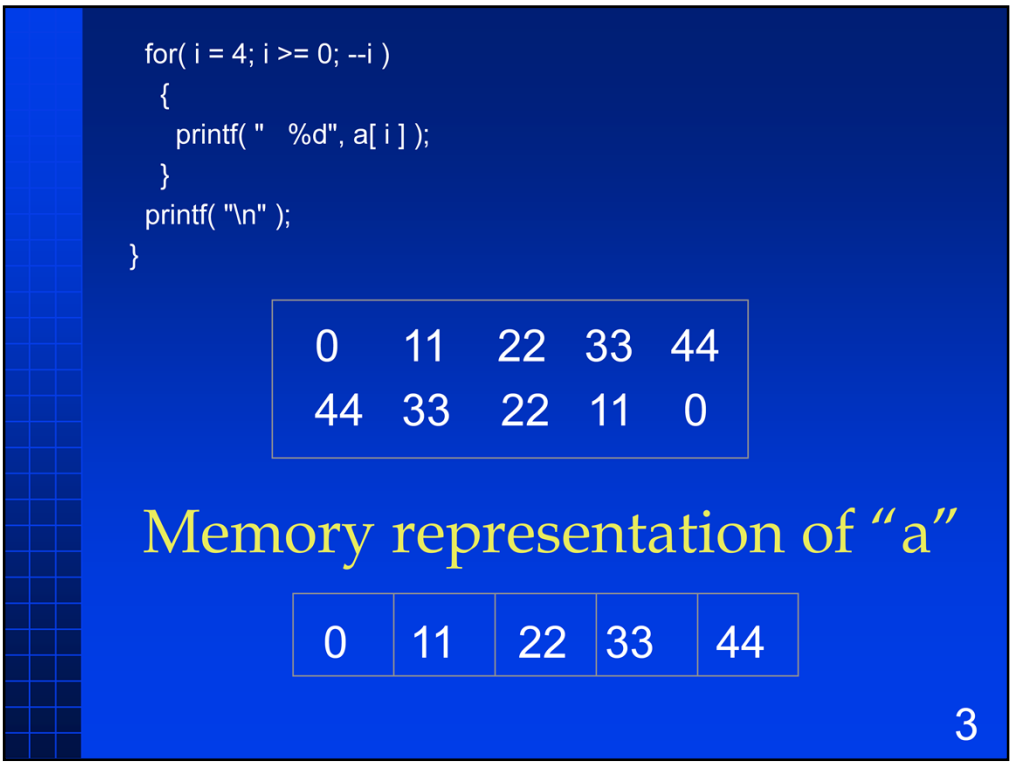



1) Array definition: 5 elements numbered 0 through 4 inclusive

2) subscripting operation: select ith element from 0. In this eg, individual elements are a[0], ..a[4]

3) array elements can be used like ordinary variables. subscripting chooses an element that is equivalent to a simple value.

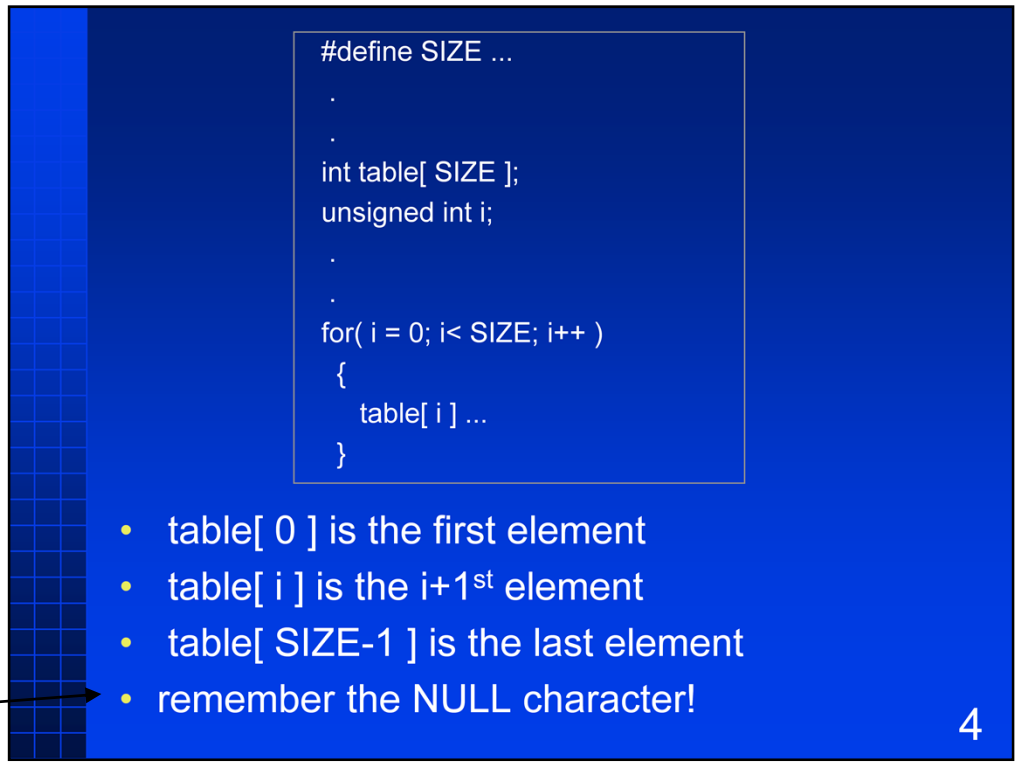
manipulate the array (access all the elements), use a for statement, use int i to select elements.




print the array backwards

program output

array in memory:  no null chars, no indication of # elements




1

Key things in array definitions:

- basetype (type of individual elements)
- # elements

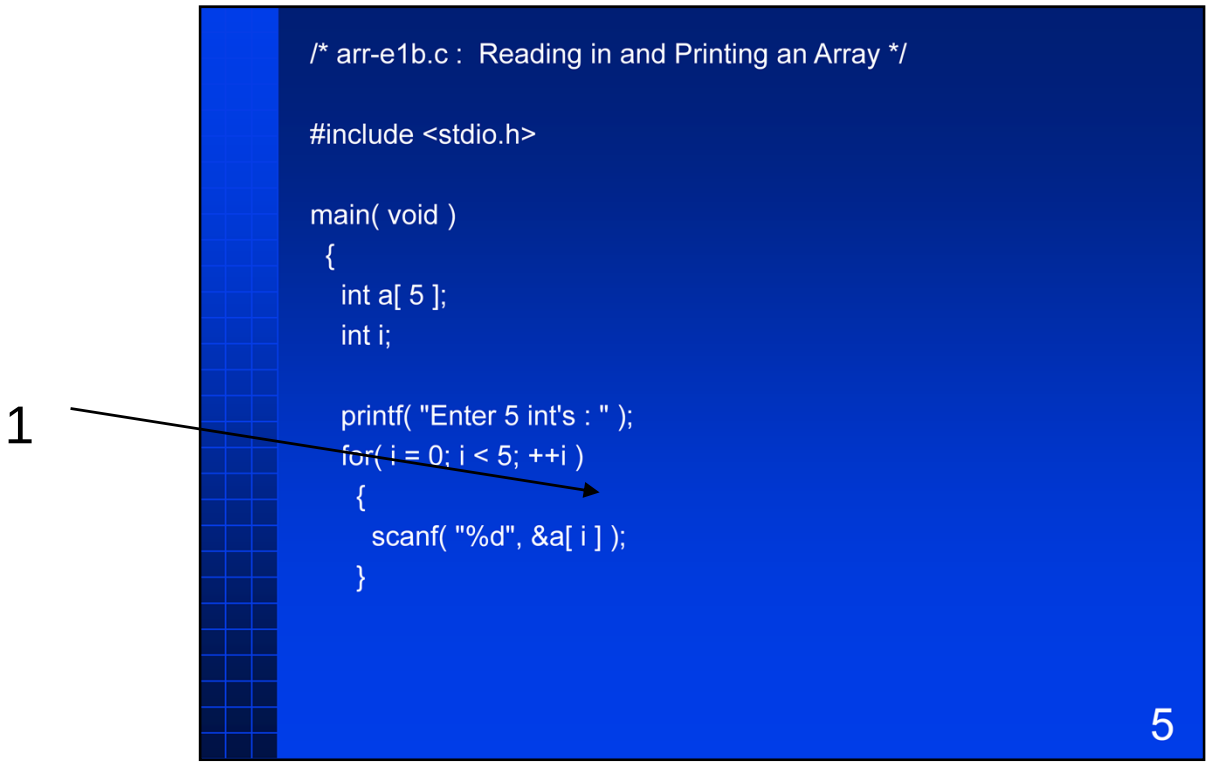
multiple dimensions: add more []. abstraction is array of array, row-major ordering.

string definitions: remember to add 1 to size to hold nullchar.

Eg: want 10 “visible” characters, need 11 characters of storage:

str[11] == str[0] .. str[9] are 10 visible characters
str[10] for the nullchar

1) table[SIZE] is past the end of the array



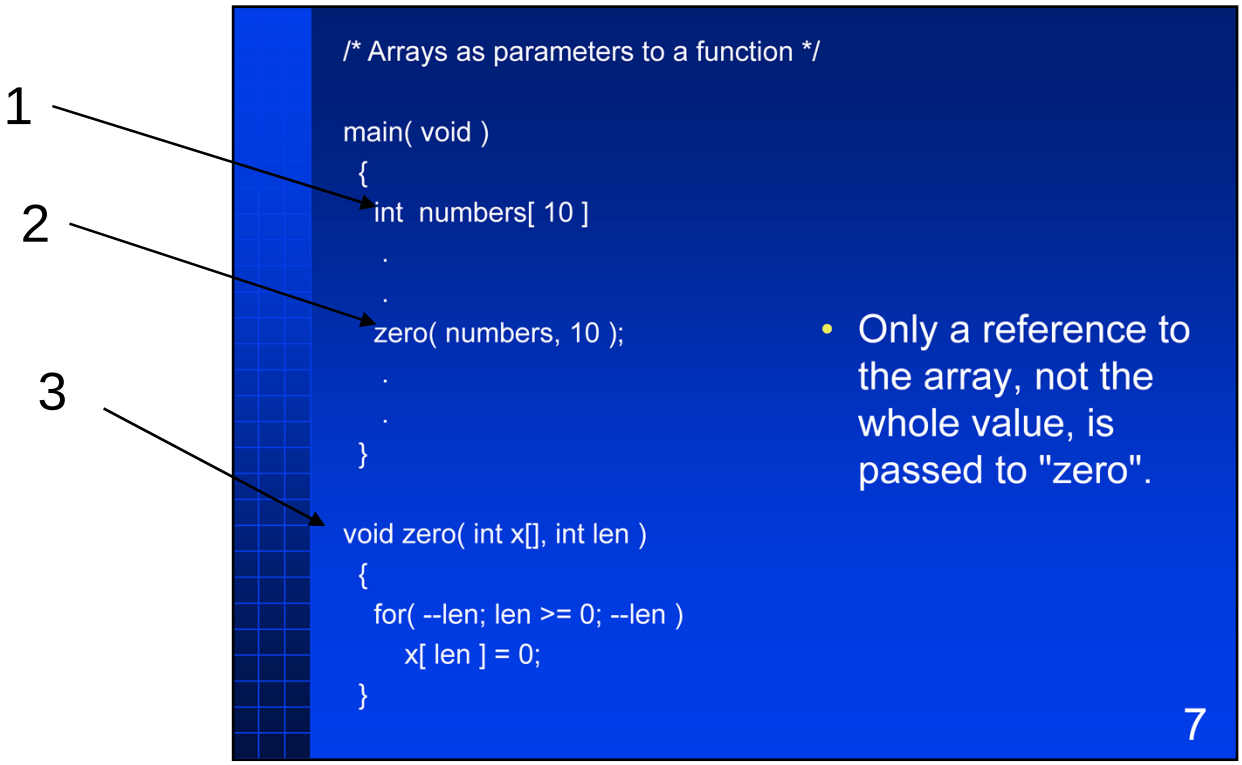

1) array elements just like simple scalar variables, so need & as before in order to address elements.

```
for( i = 0; i < 5; ++i )
  {
    printf( "   %d", a[ i ] );
  }
 printf( "\n" );

 for( i = 4; i >= 0; --i )
  {
    printf( "   %d", a[ i ] );
  }
 printf( "\n" );
}
```

| *1 3 5 7 9* |
| --- |
| 1 3 5 7 9<br>9 7 5 3 1 |



continued




Individual elements are OK, want to be able to deal with whole collection, too.

Eg want to write a function that sets all elements to zero.

1) definition of array, 10 elements numberd 0..9

2) invoke function, pass array and number of elements. have to do this because there is no implicit record of # elements.

3 ) definition of function. note empty [] in parameter list to indicate incoming array.

Array passing sends only a reference/pointer to the array. the expression "value" of an entire array is its address/reference.
Any changes to "x" in function are really changes to numbers in main.
Think of scanf of strings -- no &, redundant.

Note structure of zero. clever pre-decrements to yield elements 9..0

```
/* arr-e2.c : calculate average mark */

#include <stdio.h>

#define GRADECOUNT 5

main( void )
 {
   FILE *io;
   int number, i;
   float age, aggregate, marks[ GRADECOUNT ];
   char name[ 15 ], sex[ 2 ];
   io = fopen( "student.fil", "r" );
   if( io == NULL )
    {
      printf( "couldn't open file\n" );
      exit( 1 );
    }
```

```
    for( ; ; )
     {
       fscanf( io, "%d %s %s %d", &number, name, sex, &age );
       if( feof( io ) )
        {
          break;
        }
       for( i = 0; i < GRADECOUNT; i++ )
        {
          fscanf( io, "%f", &marks[ i ] );
        }
       aggregate = 0;
       for( i = 0; i < GRADECOUNT; i++ )
        {
          aggregate += marks[ i ];
        }
       printf( "%d %-15s %6.2f\n", number, name,
          ( aggregate / GRADECOUNT ) );
     }
    fclose( io );
}
```

```
/* arr-e3.c : numbers above average */

#include <stdio.h>

main( void )
 {
   float numbers[ 100 ], number, aggregate, average;
   int count, i;
   count = 0;
   printf( "Enter number :\n" );
   scanf( "%f", &number );
   while( number >= 0.0 )
    {
      numbers[ count ] = number;
      printf( "Enter number:\n" );
      scanf( "%f", &number );
      count++;
    }
```

```
    aggregate = 0;
    for( i = 0; i <= count; i++ )
     {
       aggregate += numbers[ i ];
     }
    average = aggregate / ( count + 1 );
    printf( "The following are above average\n" );
    for( i = 0; i <= count; i++ )
     {
       if( numbers[ i ] > average )
        {
          printf( "%10.5f\n", numbers[ i ] );
        }
     }
}
```

```
/* arr-e4.c : age count for 15 16 17 & 18 */
#include <stdio.h>
main( void )
 {
   FILE *io;
   int i, age, number, agecount[ 4 ];
   char name[ 15 ], sex[ 2 ];

   io = fopen( "student.fil", "r" );
   if( io == NULL )
    {
      printf( "couldn't open file\n" );
      exit( 1 );
    }
   for( i = 0; i <= 3; i++ )
    {
      agecount[ i ] = 0;
    }
```

```
for( ; ; )
  {
    fscanf( io, "%d%s%s%d %*f%*f%*f%*f%*f",
      &number, name, sex, &age );
    if( feof( io ) )
     {
       break;
     }
    agecount[ age - 15 ]++;
  }
 fclose( io );
 printf( "%8s %8s %8s %8s\n\n", "15", "16", "17", "18" );
 for( i = 0; i <= 3; i++ )
  {
    printf( "%8d ", agecount[ i ] );
  }
 printf("\n");
}
```

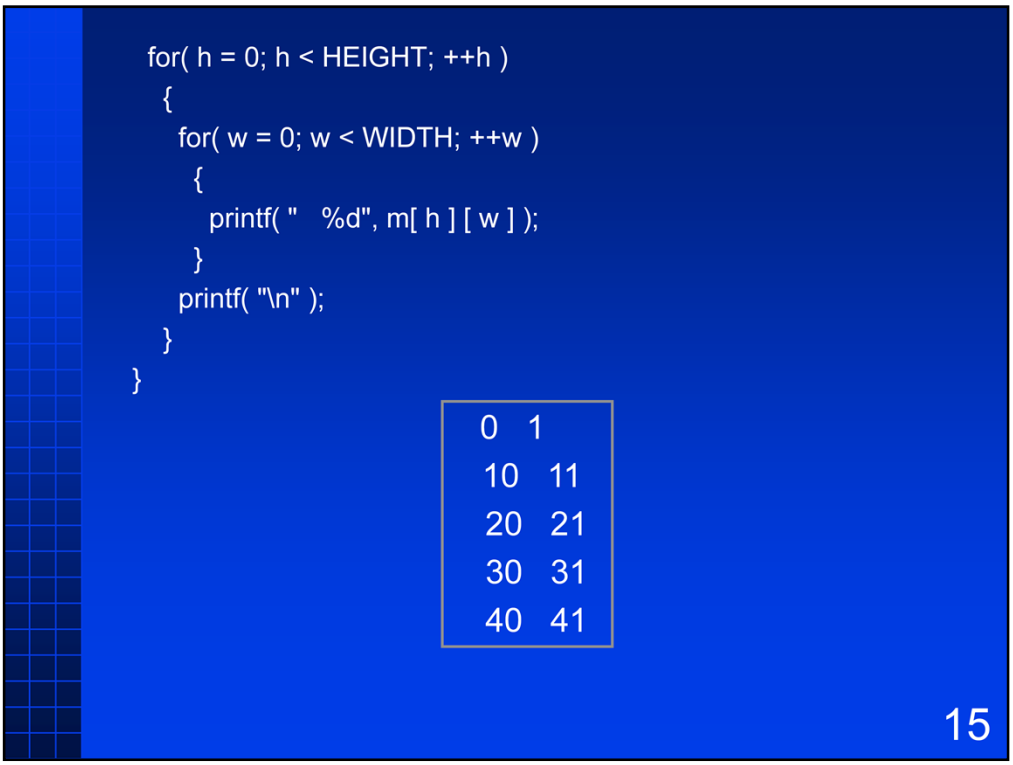
```
/* arr-e5.c :  Matrix manipulation */

#include <stdio.h>

#define HEIGHT 5
#define WIDTH 2

main( void )
 {
   int m[ HEIGHT ] [ WIDTH ];
   int h, w;

   for( h = 0; h < HEIGHT; ++h )
      for( w = 0; w < WIDTH; ++w )
        {
          m[ h ] [ w ] = 10 * h + w;
        }
```

14

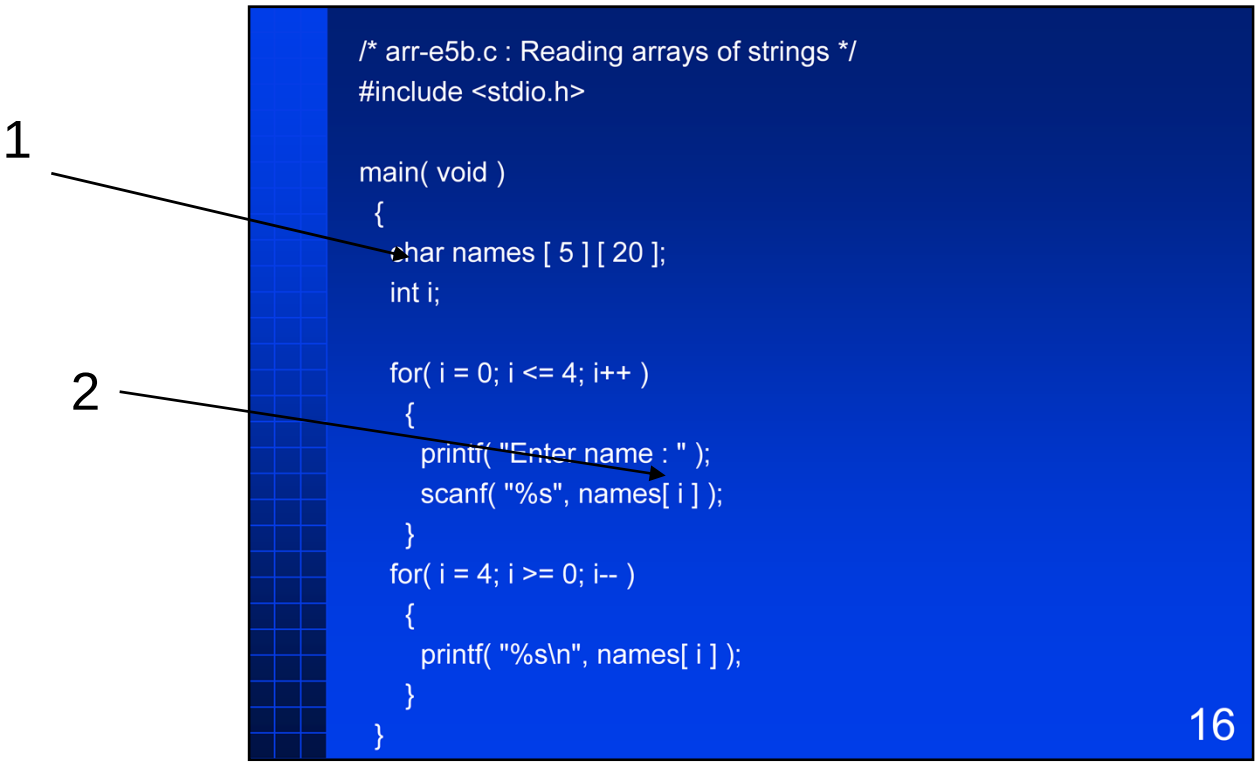
simple matrix, row-column (height==row; width == column); has HEIGHTxWIDTH individual elements

select row, then traverse columns: coding view is array of 5 elements, each element is an array itself

nested for statements common usage




display the matrix, \n after each row




1) an array of strings, really a matrix of characters, either 5, 20-char strings or 5x20 matrix of characters; 5x20=100 characters total

use the array or array concept to advantage:

2) missing subscript, names[i] select an entire array, which is a string

still don't need &, since expression yields an array.

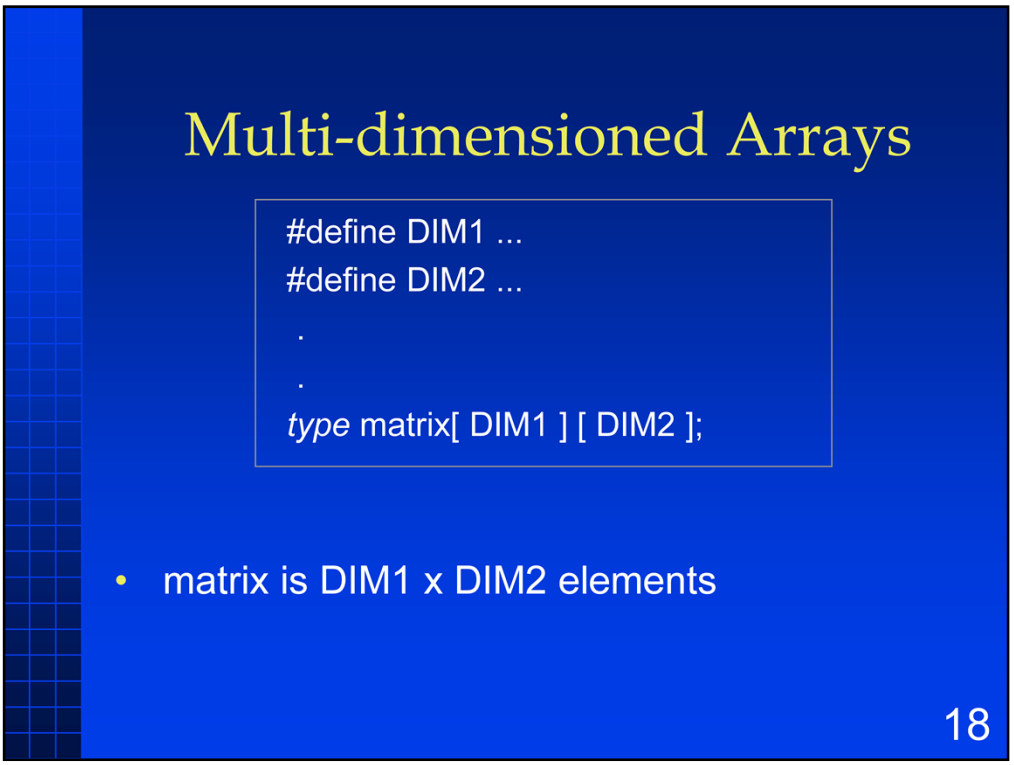
# Multi-dimensioned Arrays

```
char names [ 5 ] [ 20 ]
```

- “names[i]” refers to a row (i.e., 20-character array)
- “names[i] [j]” refers to a single character



as stated



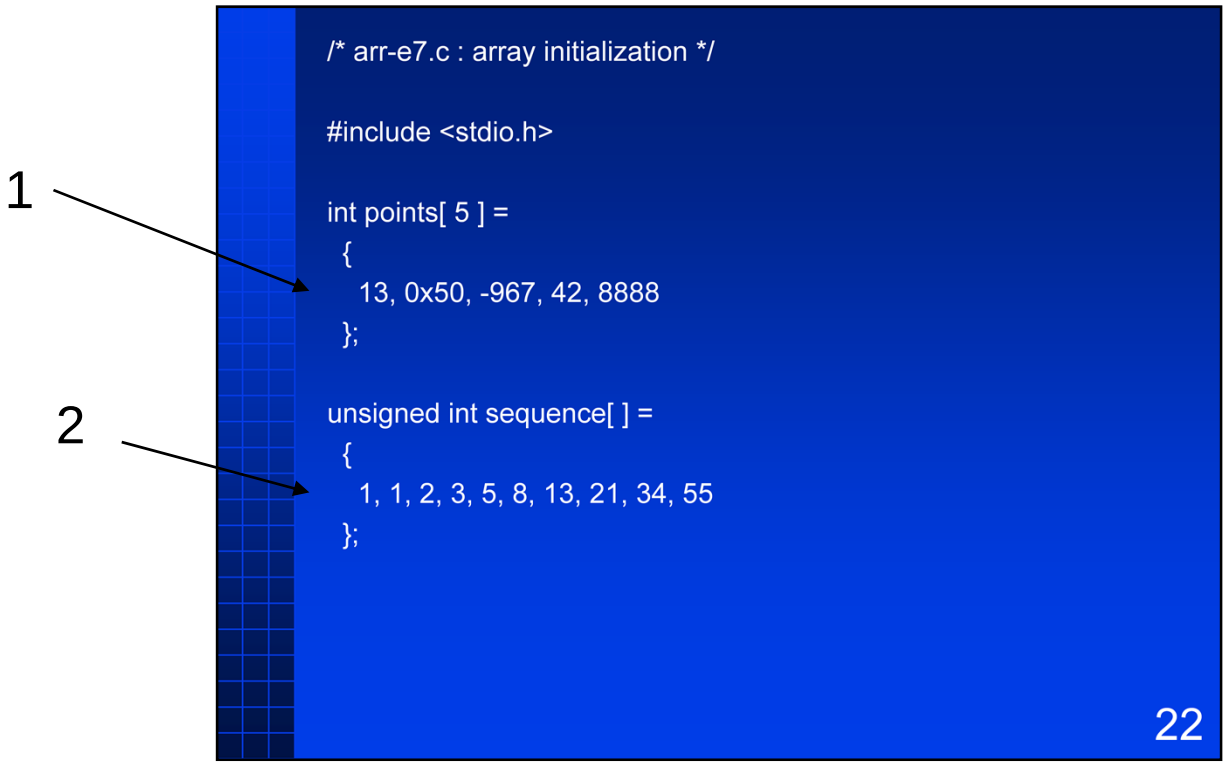

# elements is product of each dimenstion size

good coding practice to use symbolic constants as shown here, write code to use constants in for-loops etc.

```
/* arr-e6.c : Male/Female age count */
#include <stdio.h>

#define MALE 0
#define FEMALE 1

main( void )
 {
   FILE *io;
   int i, j, age, number, count[ 4 ] [ 2 ];
   char name[ 15 ], sex[ 2 ];

   io = fopen( "student.fil", "r" );
   if( io == NULL )
    {
      printf( "couldn't open file\n" );
      exit( 1 );
    }
```

```
for( i = 0; i <= 3; i++ )
    for( j = MALE; j <= FEMALE; j++ )
        count[ i ] [ j ] = 0;
for( ; ; )
 {
    fscanf( io, "%d%s%s%d %*f%*f%*f%*f%*f",
                &number, name, sex, &age );
    if( feof( io ) )
        break;
    if( strcmp( sex, "F" ) == 0 )
        count[ age - 15 ] [ FEMALE ]++;
    else
        count[ age - 15 ] [ MALE ]++;
 }
```

```
    fclose( io );
    printf( "        %8s %8s\n\n", "FEMALES",
                                   "MALES" );
    for( i = 0; i <= 3; i++ )
        printf( "age %3d %8d %8d\n", i + 15,
                count[ i ] [ FEMALE ],
                count[ i ] [ MALE ] );
}
```

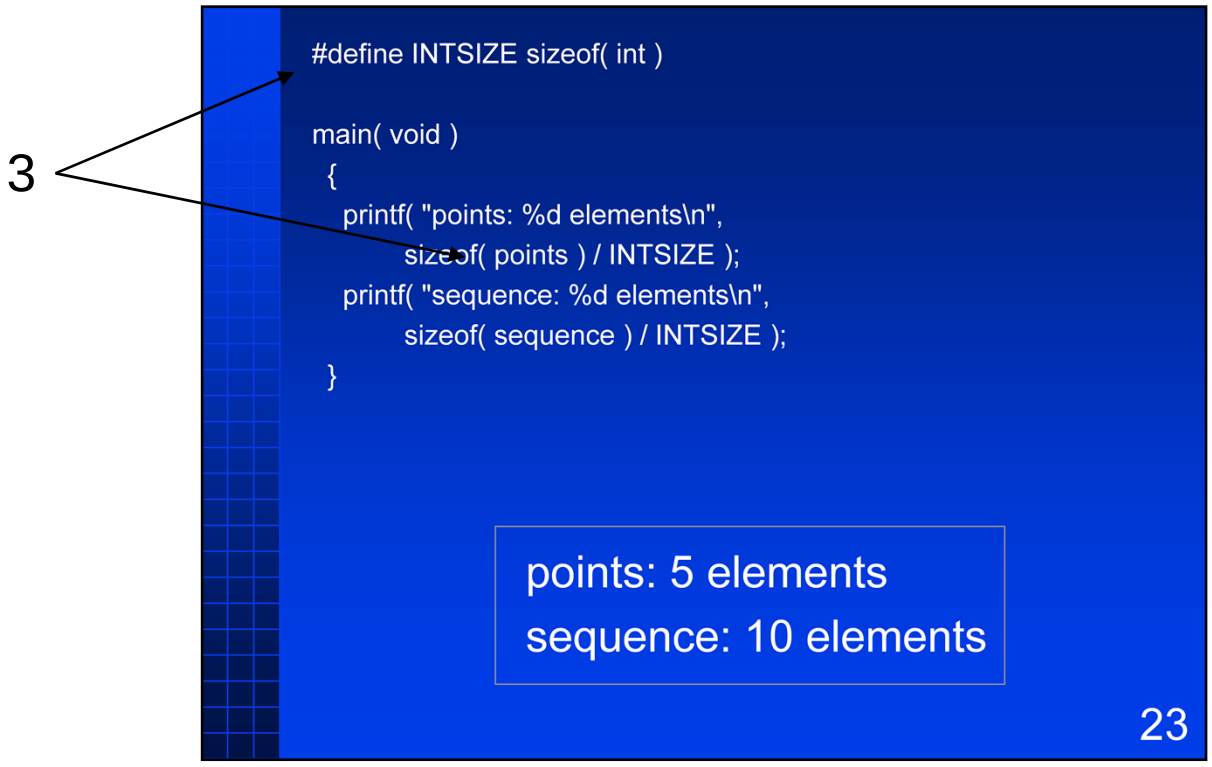



many arrays are tables of information, commonly want to be able to initialize them

C supports “structured” constants for initialization.

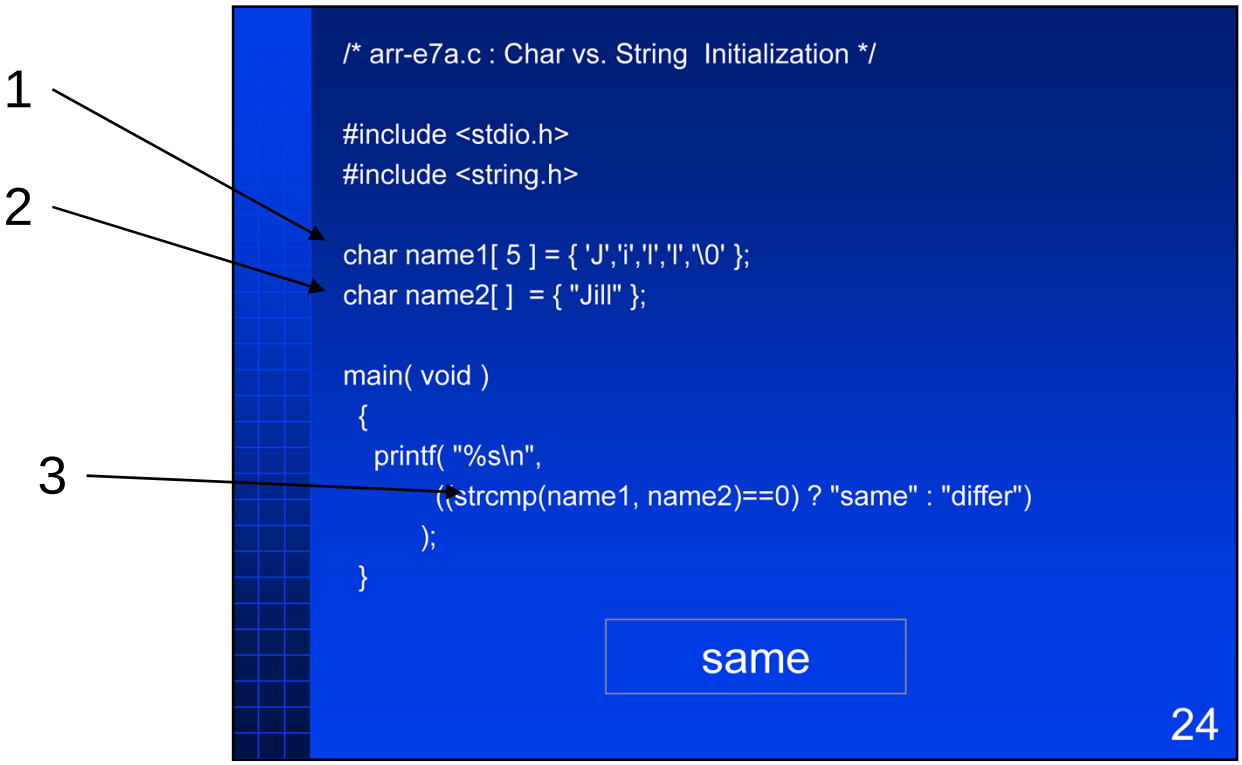
1) # elements given explicitly, followed by initial values for elements. if elements missing, some default value (zero) [this generalizes to all variables, in fact]

2) implicit # elements, # initial values determines # elements in definition.




code that computes # elements

3) sizeof  compile-time function that given #bytes storage for a type or a variable




string initialization: two styles

1) as array of characters: have to place \0 ourselves

2) using string literal, compiler inserts \0

3) ? operator


proof

```
/* arr-e8.c : Convert month number to string name */

#include <stdio.h>

static char Months[ 12 ] [ 10 ] =
 {
    "January",
    "February",
    "March",
    "April",
    "May",
    "June",
    "July",
    "August",
    "September",
    "October",
    "November",
    "December"
 };
```

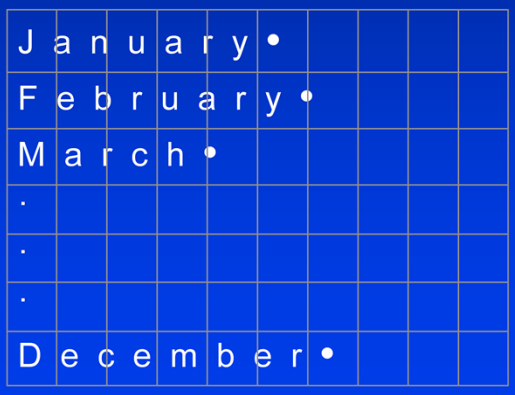


initializing an array of string == initializing a matrix of characters.

use representation that makes sense:  12 rows of 10-char strings

cannot omit second dimension in definition here: must know big strings are (could omit first, compiler can count # strings), due to matrix idea, rectangular block of characters, compiler will pad out to 10-char each.

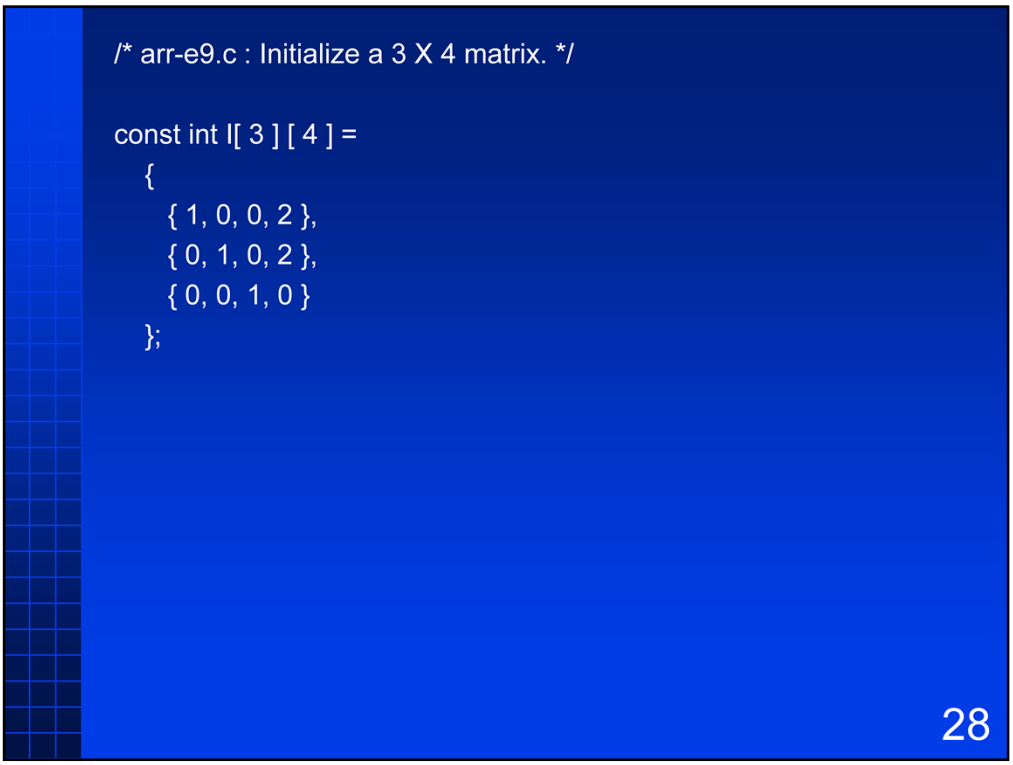
```
main( void )  {
   int day, month, year;

   scanf( "%d %d %d", &day, &month, &year );
   printf( "%s %d, %d.\n",
         Months[ month-1 ], day, year );
}
```






storage representation of array

bad program, doesn't bounds-check




matrix initialization: can use braces to construct an initializer that follows the structure of the array.

here, 3 rows of 4 integers each

```
/* arr-e10.c : Matrix Add  c ij =  a ij + b ij */

static void madd( int x[ 3 ] [ 3 ],
                  int y[ 3 ] [ 3 ],
                  int z[ 3 ] [ 3 ] );
static void readmatrix( int x[ 3 ] [ 3 ] );
static void printmatrix( int x[ 3 ] [ 3 ] );

main( void )
 {
   int a[ 3 ] [ 3 ];
   int b[ 3 ] [ 3 ];
   int c[ 3 ] [ 3 ];

   readmatrix( a );
   readmatrix( b );
   madd( a, b, c );
   printmatrix( a );
   printf( "+\n" );
   printmatrix( b );
   printf( "=\n" );
   printmatrix( c );
 }
```

```
static void madd( int x[ 3 ] [ 3 ],
                  int y[ 3 ] [ 3 ],
                  int z[ 3 ] [ 3 ] )
 {
   int i, j;

   for( i = 0; i < 3; ++i )
     {
       for( j = 0; j < 3; ++j )
         {
           z[ i ] [ j ] = x[ i ] [ j ] +
                          y[ i ] [ j ];
         }
     }
 }
```

```
static void readmatrix( int x[ 3 ] [ 3 ] )
 {
   int i,j;

   printf( "Enter a 3 by 3 matrix\n" );
   for( i = 0; i < 3; ++i )
    {
      for( j = 0; j < 3; ++j )
       {
         scanf( "%d", &x[ i ] [ j ] );
       }
    }
 }
```

```
static void printmatrix( int x[ 3 ] [ 3 ] )
 {
   int i, j;

   for( i = 0; i < 3; ++i )
    {
      for( j = 0; j < 3; ++j )
       {
         printf( "%4d", x[ i ] [ j ] );
       }
      printf( "\n" );
    }
 }
```