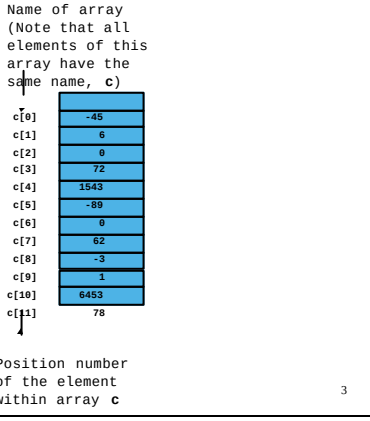
Arrays

Arrays

- Array
  - Group of consecutive memory locations
  - Same name and type
- To refer to an element, specify
  - Array name
  - Position number
- Format:

  arrayname **[** *position number* **]**
  - First element at position **0**
  - **n** element array named **c**:
    - **c[0]**, **c[1]**…**c[n – 1]**



Array Elements

- Array elements are like normal variables

```
c[0] =  3;
printf( "%d", c[0] );
```

- We can perform operations in subscript.

  e.g. If **x** equals **3**

```
c[5-2] == c[3] == c[x]
c[x+1] == c[4]
c[x-1] == c[2]
```

Declaring Arrays

- When declaring arrays, specify
  - Name
  - Type of array
  - Number of elements
    ```
    arrayType arrayName[ numberOfElements ];
    ```
  - Examples:
    ```
    int c[10];
    float myArray[3284];
    ```
- Declaring multiple arrays of same type
  - Format similar to regular variables
  - Example:
    ```
    int b[100], x[27];
    ```

5

Examples Using Arrays

- Initializers
  ```
  int n[5] = { 1, 2, 3, 4, 5 };
  ```
  - If not enough initializers, rightmost elements become **0**
    ```
    int n[5] = { 0 }
    ```
    - All elements 0
  - If too many a syntax error is produced syntax error
  - C arrays have no bounds checking
- If size omitted, initializers determine it
  ```
  int n[ ] = { 1, 2, 3, 4, 5 };
  ```
  - 5 initializers, therefore 5 element array

6

Initializing an Array

```
#include <stdio.h>

int main(void)
{
  int n[100], i;
  for (i=0; i < 100; i++)
      n[i] = i;

  for (i=0; i < 100; i++)
      printf("Element %d has value %d.\n",i, n[i]);
  return 1;
}
```

7

Examples

- Reading values into an array

```
int i, x[100];

for (i=0; i < 100; i=i+1) {
   printf("Enter an integer: ");
   scanf("%d",&x[i]);
}
```

- Summing up all elements in an array

```
int sum = 0;
for (i=0; i<=99; i=i+1)
   sum = sum + x[i];
```

8

Examples (contd.)

- Shifting the elements of an array to the left.

```
/* store the value of the first element in a
 * temporary variable
 */
temp = x[0];

for (i=0; i < 99; i=i+1)
   x[i] = x[i+1];

//The value stored in temp is going to be
the value of the last element:
x[99] = temp;
```


Examples

- Finding the location of a given value (`item`) in an array.

```
i = 0;
while ((i<100) && (x[i] != item))
   i = i + 1;

if (i == 100)
   loc = -1; // not found
else
   loc = i;  // found in location i
```



```
1  /* Histogram printing program */
2
3  #include <stdio.h>
4  #define SIZE 10
5
6  int main()
7  {
8     int n[ SIZE ] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
9     int i, j;
10
11    printf( "%s%13s%17s\n", "Element", "Value", "Histogram" );
12
13    for ( i = 0; i <= SIZE - 1; i++ ) {
14       printf( "%7d%13d        ", i, n[i]) ;
15
16       for ( j = 1; j <= n[ i ]; j++ )   /* print one bar */
17          printf( "%c", '*' );
18
19       printf( "\n" );
20    }
21
22    return 0;
23 }
```


Program Output

```
Element        Value        Histogram
      0           19        ********************
      1            3        ***
      2           15        ***************
      3            7        *******
      4           11        ***********
      5            9        *********
      6           13        *************
      7            5        *****
      8           17        *****************
      9            1        *
```

Passing Arrays to Functions

- Passing arrays
  - To pass an array argument to a function, specify the name of the array without any brackets

```
int myArray[ 24 ];  //declaration in main
myFunction( myArray, 24 ); //calling the function
```

    - Array size is usually passed to function
  - Arrays passed call-by-reference
  - Name of array is address of first element
  - Function knows where the array is stored
    - Modifies original memory locations
- Passing array elements
  - Passed by call-by-value
  - Pass subscripted name (i.e., **`myArray[ 3 ]`**) to function

Passing Arrays to Functions

- Function prototype

```
void modifyArray( int b[], int arraySize );
```

- Parameter names optional in prototype
  - **`int b[]`** could be written **`int []`**
  - **`int arraySize`** could be simply **`int`**

Example

```
int sum(int a[], int n)
{
   int j, s=0;
   for (j=0; j <n ; j++)
        s = s+ a[j];
   return s;
}
```

- *Note*: a[] is a notational convenience. In fact

```
int a[]   º   int *a
```

- Calling the function:

```
int total, x[100];

total = sum(x, 100);
total = sum(x, 88);
total = sum(&x[5], 50);
```



```
1  /* Passing arrays and individual array elements to functions */
2
3  #include <stdio.h>
4  #define SIZE 5
5
6  void modifyArray( int [], int );  /* appears strange */
7  void modifyElement( int );
8
9  int main()
10 {
11    int a[ SIZE ] = { 0, 1, 2, 3, 4 }, i;
12
13    printf( "Effects of passing entire array call "
14            "by reference:\n\nThe values of the "
15            "original array are:\n" );
16
17    for ( i = 0; i <= SIZE - 1; i++ )
18       printf( "%3d", a[ i ] );
19
20    printf( "\n" );
21    modifyArray( a, SIZE );  /* passed call by reference */
22    printf( "The values of the modified array are:\n" );
23
24    for ( i = 0; i <= SIZE - 1; i++ )
25       printf( "%3d", a[ i ] );
26
27    printf( "\n\n\nEffects of passing array element call "
28            "by value:\n\nThe value of a[3] is %d\n", a[ 3 ] );
29    modifyElement( a[ 3 ] );
30    printf( "The value of a[ 3 ] is %d\n", a[ 3 ] );
31    return 0;
32 }
```

Entire array 's passed call-by-reference, and can be modified

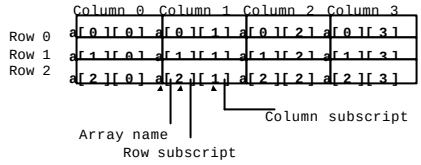
Array element is passed call-by-value, and cannot be modified

```
33
34 void modifyArray( int b[], int size )
35 {
36    int j;
37
38    for ( j = 0; j <= size - 1; j++ )
39       b[ j ] *= 2;
40 }
41
42 void modifyElement( int e )
43 {
44    printf( "Value in modifyElement is %d\n", e *= 2 );
45 }
```

```
Effects of passing entire array call by reference:

The values of the original array are:
  0  1  2  3  4
The values of the modified array are:
  0  2  4  6  8

Effects of passing array element call by value:


The value of a[3] is 6
Value in modifyElement is 12
The value of a[3] is 6
```


Two Dimensional Arrays

- Multiple subscripted arrays
  - Tables with rows and columns (**m** by **n** array)
  - Like matrices: specify row, then column



Two Dimensional Arrays

- Initialization
  - `int b[ 2 ][ 2 ] = { { 1, 2 }, { 3, 4 } };`

  - Initializers grouped by row in braces
  - If not enough, unspecified elements set to zero
    `int b[ 2 ][ 2 ] = { { 1 }, { 3, 4 } };`

- Referencing elements
  - Specify row, then column
    `printf( "%d", b[ 0 ][ 1 ] );`

Examples

- Reading values into a two-dimensional array:

```
int a[10][20];
for (row=0; row < 10; row = row+1){
   for(col=0; col < 20; col = col+1) {
      printf("Enter a number: ");
      scanf("%d",&a[row][col]);
   }
}
```

```c
/* Two-dimensional array example
   */
#include <stdio.h>
#define STUDENTS 3
#define EXAMS 4

int minimum( int [][ EXAMS ], int, int );
int maximum( int [][ EXAMS ], int, int );
double average( int [], int );
void printArray( int [][ EXAMS ], int, int);

int main()
{
   int student;
   int studentGrades[ STUDENTS ][ EXAMS ] =
      { { 77, 68, 86, 73 },
        { 96, 87, 89, 78 },
        { 70, 90, 86, 81 } };

   printf( "The array is:\n" );
   printArray( studentGrades, STUDENTS, EXAMS );
   printf( "\n\nLowest grade: %d\nHighest grade: %d\n",
           minimum( studentGrades, STUDENTS, EXAMS ),
           maximum( studentGrades, STUDENTS, EXAMS ) );

   for ( student = 0; student <= STUDENTS - 1; student++ )
      printf( "The average grade for student %d is %.2f\n",
              student,
              average( studentGrades[ student ], EXAMS ) );

   return 0;
}
```

Each row is a particular student, each column is the grades on the exam.


```c

/* Find the minimum grade */
int minimum( int grades[][ EXAMS ],
             int pupils, int tests )
{
   int i, j, lowGrade = 100;

   for ( i = 0; i <= pupils - 1; i++ )
      for ( j = 0; j <= tests - 1; j++ )
         if ( grades[ i ][ j ] < lowGrade )
            lowGrade = grades[ i ][ j ];

   return lowGrade;
}

/* Find the maximum grade */
int maximum( int grades[][ EXAMS ],
             int pupils, int tests )
{
   int i, j, highGrade = 0;

   for ( i = 0; i <= pupils - 1; i++ )
      for ( j = 0; j <= tests - 1; j++ )
         if ( grades[ i ][ j ] > highGrade )
            highGrade = grades[ i ][ j ];

   return highGrade;
}

/* Determine the average grade for a particular exam */
double average( int setOfGrades[], int tests )
{
```



```c
   int i, total = 0;

   for ( i = 0; i <= tests - 1; i++ )
      total += setOfGrades[ i ];

   return ( double ) total / tests;
}

/* Print the array */
void printArray( int grades[][ EXAMS ],
                 int pupils, int tests )
{
   int i, j;

   printf( "                [0]  [1]  [2]  [3]" );

   for ( i = 0; i <= pupils - 1; i++ ) {
      printf( "\nstudentGrades[%d] ", i );

      for ( j = 0; j <= tests - 1; j++ )
         printf( "%-5d", grades[ i ][ j ] );
   }
}
```



```
The array is:
                [0]  [1]  [2]  [3]
studentGrades[0] 77   68   86   73
studentGrades[1] 96   87   89   78
studentGrades[2] 70   90   86   81

Lowest grade: 68
Highest grade: 96
The average grade for student 0 is 76.00
The average grade for student 1 is 87.50
The average grade for student 2 is 81.75
```

Program Output