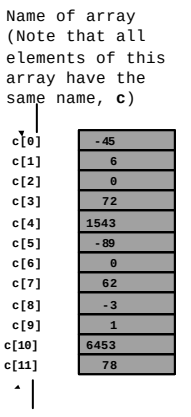
Arrays

Array

- 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]**



Using Arrays

- Array elements are like normal variables

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

  - Perform operations in subscript. If **x** equals **3**

```
c[ 5 - 2 ] == c[ 3 ] == c[ x ]
```

- There are no built-in operators for arrays

```
int i,a[5],b[5],c[5];
a=1;                    /* error */
b=2;                    /* error */
for(i=0;i<5;i++)
{ a[i]=1; b[i]=2; }     /* ok */
c=a+b;                  /* error */
for(i=0;i<5;i++)
  c[i]=a[i]+b[i];       /* ok */
```


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 ];
```

Examples Using Arrays

- Initializers

```
int n[ 5 ] = { 1, 2, 3, 4, 5 };
```

  - If not enough initializers, rightmost elements become **0**

```
int n[ 5 ] = { 0 }
```

  - If too many a syntax error is produced
  - C arrays have no bounds checking
- If size omitted, initializers determine it

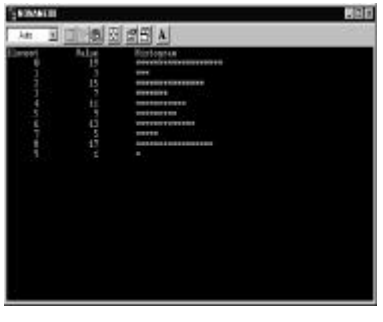
```
int n[ ] = { 1, 2, 3, 4, 5 };
```

  - 5 initializers, therefore 5 element array

Example 5A

```
/* 5A: Histogram printing program */
#include <stdio.h>
#include <conio.h>
#define SIZE 10
int main()
{
  int n[ SIZE ] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
  int i, j;

  printf( "%s%13s%17s\n", "Element", "Value", "Histogram" );
  for ( i = 0; i <= SIZE - 1; i++ ) {
    printf( "%7d%13d          ", i, n[ i ]) ;

    for ( j = 1; j <= n[ i ]; j++ )   /* print one bar */
      printf( "%c", '*' );

    printf( "\n" );
  }
  getch();
  return 0;
}
```


Output Result



Examples Using Arrays

- Character arrays
  - String **"first"** is really a static array of characters
  - Character arrays can be initialized using string literals

```
char string1[] = "first";
```

    - Null character **'\0'** terminates strings
    - **string1** actually has 6 elements
      - It is equivalent to

```
char string1[] = { 'f', 'i', 'r', 's', 't', '\0' };
```

  - Can access individual characters

```
string1[ 3 ] is character 's'
```

  - Array name is address of array, so & not needed for scanf

```
scanf( "%s", string2 );
```

    - Reads characters until whitespace encountered
    - Can write beyond end of array, be careful

Examples Using Arrays

- Indices must be integers
  - int i, j, a[10];
  - float r = 1.0;
  - a[-1] = 1; /* error */
  - a[10] = 1; /* error */
  - a[15] = 1; /* error */
  - a[r] = 1; /* error */
  - j = r; /* j=1 */
  - a[j] = 1; /* okay */

Example 5A

```
/* EB: Treating character arrays as strings */
#include <stdio.h>
#include <conio.h>
int main()
{
   char string1[ 20 ], string2[] = "string literal";
   int i;

   printf(" Enter a string: ");
   scanf( "%s", string1 );
   printf( "string1 is: %s\nstring2 is: %s\n"
         "string1 with spaces between characters is: \n",
         string1, string2 );

   for ( i = 0; string1[ i ] != '0'; i++ )
     printf( "%c ", string1[ i ] );
   printf( "\n" );
   getch();
   return 0;
}
```


Output Result

Passing Arrays to Functions

- We prototype a function which accepts an array like this:
  - void process_array (int [ ]);
  - int calc_array (char[ ]);
- And write the function like this:
  - void process_array (int all_nums[ ]) {
  - all_nums[1]= 3;
  - …
  - }
- And call the function like this:
  - int some_numbers [100];
  - process_array(some_numbers);
- Note that we CAN'T return an array from a function (but see later).

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 ];
myFunction( myArray, 24 );
```

    - Array size 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

Pass by reference/Pass by value

- Normally when we send a variable to a function we make a COPY of the variable.
- This is called *pass by value*. A value is passed and this copy of the variable arrives at the function.
- Sometimes, like in scanf we want to change the variable inside the function.
- In this case we need to do something different: *pass by reference*.
- This is what the & character is for.

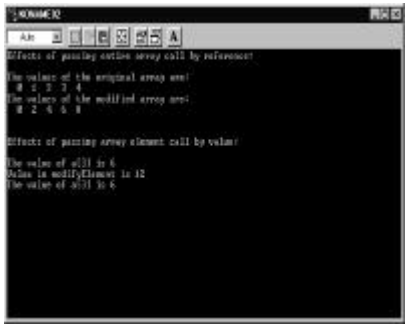
Example 5C

```
/* 5C
Passing arrays and individual array elements to functions */
#include <stdio.h>
#include <conio.h>
#define SIZE 5

void modifyArray( int [], int );  /* appears strange */
void modifyElement( int );
int main()
{
   int a[ SIZE ] = { 0, 1, 2, 3, 4 }, i;
   printf( "Effects of passing entire array call "
           "by reference:\n\nThe values of the "
           "original array are:\n" );

   for ( i = 0; i <= SIZE - 1; i++ )
      printf( "%3d", a[ i ] );
   printf( "\n" );
   modifyArray( a, SIZE );  /* passed call by reference */
   printf( "The values of the modified array are:\n" );
```


Example 5C

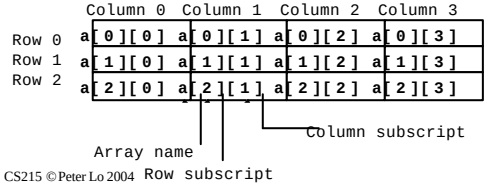
```
   for ( i = 0; i <= SIZE - 1; i++ )
      printf( "%3d", a[ i ] );

   printf( "\n\n\nEffects of passing array element call "
           "by value:\n\nThe value of a[3] is %d\n", a[ 3 ] );
   modifyElement( a[3] );
   printf( "The value of a[3] is %d\n", a[ 3 ] );
   getch();
   return 0;
}
void modifyArray( int b[], int size )
{
   int j;
   for ( j = 0; j <= size - 1; j++ )
      b[ j ] *= 2;
}
void modifyElement( int e )
{
   printf( "Value in modifyElement is %d\n", e *= 2 );
}
```

Output Result



Multiple-Subscripted Arrays

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



Multiple-Subscripted Arrays

- Initialization
  - `int b[2][2]={{1,2},{3,4}};`
  - Initializers grouped by row in braces
  - If not enough, unspecified elements set to 0/null

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

- Referencing elements
  - Specify row, then column

`printf("%d",b[0][1]);`

Example 5D

```
/* 5 D
Double-subscripted array example */
#include<stdio.h>
#include<conio.h>
#define STUDENTS 3
#define EXAMS 4
int minimum( const int [][ EXAMS ], int, int );
int maximum( const int [][ EXAMS ], int, int );
double average( const int [], int );
void printArray ( const int [][ EXAMS ], int, int );

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

Example 5D

```
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 ) );

getch ();
return 0;
}
```


Example 5D

```
/* Find the minimum grade */
int minimum( const 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( const 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;
}
```


Example 5D

```
/* Determine the average grade for a particular exam */
double average( const int setOfGrades[],  int tests )
{  int  i, total = 0;
   for ( i = 0; i <= tests - 1; i++ )
     total += setOfGrades[ i ];
   return ( double ) total / tests;
}
/* Print the array */
void printArray ( const 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 ] );
   }
}
```


Output Result