Computer Programming Using C
COP 3275 - Summer 2017

Lecture 12: Array (cont.)

By Ahmed E. Khaled
Recap to previous lecture!

- Type Conversion
- Array
Type Conversion

• When operands of different types are mixed in expressions, the compiler may have to generate instructions that change the types of some operands so that hardware will be able to evaluate the expression.

• Because the compiler handles these conversions automatically, without the programmer’s involvement, they’re known as *implicit conversions*.

• C also allows the programmer to perform *explicit conversions*, using the *cast operator*. 
One-Dimensional Arrays

• An array is a data structure containing a number of data values, all of which have the same type.
• These values, known as elements, can be individually selected by their position within the array.
• The simplest kind of array has just one dimension (1D).

• Array A :

| [0] | [1] | [2] | ...... | [n-1] |
Array size

```c
int abc [10];
```

- The elements of an array of length $n$ are indexed from 0 to $n - 1$.
- The elements of abc are:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• Develop a program that calculate the summation of array elements, array size is 50.

    int i;
    int sum = 0;
    for (i = 0; i < N; i++){
        sum += a[i];
    }
• C doesn’t require that subscript bounds be checked; if a subscript goes out of range, the program’s behavior is undefined.

• A common mistake: forgetting that an array with $n$ elements is indexed from 0 to $n - 1$, not 1 to $n$:

```c
int a[10], i;
for (i = 1; i <= 10; i++)
    a[i] = 0;
```
• Develop a program that prompts the user to enter a series of numbers, then *prints* the numbers in *reverse order*:

Enter 10 numbers: 34 82 49 102 7 94 23 11 50 31
In reverse order: 31 50 11 23 94 7 102 49 82 34
#include <stdio.h>

#define N 10

int main(void) {
    int a[N], i;

    printf("Enter %d numbers: ", N);
    for (i = 0; i < N; i++)
        scanf("%d", &a[i]);

    printf("In reverse order:");
    for (i = N - 1; i >= 0; i--)
        printf(" %d", a[i]);
    printf("\n");

    return 0;
}
Array Initialization

• An array, like any other variable, can be given an initial value at the time it’s declared.

• The most common form of array initializer is a list of constant expressions enclosed in braces and separated by commas:

  \[
  \text{int } a[10] = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\};
  \]

• If the initializer is shorter than the array, the remaining elements of the array are given the value 0:

  \[
  \text{int } a[10] = \{1, 2, 3, 4, 5, 6\};
  \]
  \[
  /* \text{initial value of } a \text{ is } \{1, 2, 3, 4, 5, 6, 0, 0, 0, 0\} */
  \]
• If an initializer is present, the length of the array may be omitted:
  
  ```
  int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  ```

• The compiler uses the length of the initializer to determine how long the array is.
• Develop a program that prints all negative elements in an array.

• Develop a program that adds two arrays of the same length and store the result in a third array.
Using the `sizeof` Operator

• The `sizeof` operator can determine the size of an array (in bytes).

• If `a` is an array of 10 integers, then `sizeof(a)` is typically 40 (assuming that each integer requires four bytes).

• We can also use `sizeof` to measure the size of an array element, such as `a[0]`.

• Dividing the array size by the element size gives the length of the array:

\[
\text{sizeof(a) / sizeof(a[0])}
\]
• Some programmers use this expression when the length of the array is needed.

• A loop that clears the array a:

```c
for (i = 0; i < sizeof(a)/sizeof(a[0]); i++)
{
    a[i] = 0;
}
```
• Some compilers produce a warning message for this expression.

• The variable $i$ probably has type `int` (*signed type*), whereas `sizeof` produces a value of type an *unsigned type*.

• To avoid a warning, we can add a *cast* that converts the expression to a signed integer:

```c
for (i = 0; i < (int)(sizeof(a)/sizeof(a[0])); i++)
    a[i] = 0;
```
Multidimensional Arrays

- An array may have any number of dimensions.
- The following declaration creates a two-dimensional array (a \textit{matrix}, in mathematical terminology):
  
  ```
  int m[5][9];
  ```
- \( m \) has 5 rows and 9 columns. Both rows and columns are indexed from 0:
Multidimensional Arrays

• To access the element of \( m \) in row \( i \), column \( j \), we must write \( m[i][j] \).

• The expression \( m[i] \) designates row \( i \) of \( m \), and \( m[i][j] \) then selects element \( j \) in this row.

• Resist the temptation to write \( m[i,j] \) instead of \( m[i][j] \).

• C treats the comma as an operator in this context, so \( m[i,j] \) is the same as \( m[j] \).
Multidimensional Arrays

- Although we visualize two-dimensional arrays as tables, that’s not the way they’re actually stored in computer memory.
- C stores arrays in row-major order, with row 0 first, then row 1, and so forth.
- How the m array is stored:
Multidimensional Arrays

- Nested *for* loops are ideal for processing multidimensional arrays.
- Consider the problem of initializing an array for use as an identity matrix. A pair of nested *for* loops is perfect:

```c
#define N 10

double ident[N][N];
int row, col;

for (row = 0; row < N; row++)
    for (col = 0; col < N; col++)
        if (row == col)
            ident[row][col] = 1.0;
        else
            ident[row][col] = 0.0;
```