CS 210 **Oct. 4, 2016**
Day 11

Ch. 7 – Array Pointers

Basic Terminology

- data structure
  - a composite of related data items stored under the same name

- array
  - a collection of data items of the same type

In mathematics we see the notation:

$y = \sum_{i=0}^{5} x_i$ which is taken to mean $y = x_0 + x_1 + x_2 + x_3 + x_4 + x_5$

The variable name *x* is a single name which is used to stand for multiple variable by use of a subscript.

In C we write a subscript using brackets as in x[0] + x[1] + …

The number inside the brackets is the *subscript* and is always in int or an expression that evaluates to an int. This allows us to calculate the name of a variable.

Declaring and Referencing Arrays

- array element
  - a data item that is part of an array
- subscripted variable
  - a variable followed by a subscript in brackets, designating an array element
- array subscript
  - a value or expression enclosed in brackets after the array name, specifying which array element to access

```
double x[8];
```

Array x

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
16.0	12.0	6.0	8.0	2.5	12.0	14.0	-54.5

Note that
1. The first subscript is always 0 as in x[0].
2. In the array declaration double x[8] give the array type (double) and the size of the array (8).
3. The array size must be a constant.
4. The last subscript is always one less than the array size.

TABLE 7.1 Statements That Manipulate Array x

Statement	Explanation
`printf("%.1f", x[0]);`	Displays the value of `x[0]`, which is `16.0`.
`x[3] = 25.0;`	Stores the value `25.0` in `x[3]`.
`sum = x[0] + x[1];`	Stores the sum of `x[0]` and `x[1]`, which is `28.0` in the variable `sum`.
`sum += x[2];`	Adds `x[2]` to `sum`. The new `sum` is `34.0`.
`x[3] += 1.0;`	Adds `1.0` to `x[3]`. The new `x[3]` is `26.0`.
`x[2] = x[0] + x[1];`	Stores the sum of `x[0]` and `x[1]` in `x[2]`. The new `x[2]` is `28.0`.

Initializing an array

```
int x[] = {1, 2, 3, 4, 5};  //compiler fills in the right size
char vowels[] = {'a', 'e', 'i', 'o', 'u'};
```

You can also store a string in an array. In fact, a string is just an array of char where the last character is 0 (NULL).

```
char hello[] = "Hello Mom!";
```

In memory this becomes:

[10]	0
[9]	'!'
[8]	'm'
[7]	'o'
[6]	'M'
[5]	' '
[4]	'o'
[3]	'l'
[2]	'l'
[1]	'e'
hello[0]	'H'

We will do strings in Ch 8

Sometimes arrays are initialized using loops:

```
int i;
int x[100];
for(i=0;i<100;i++)
   x[i] = i*i;
```

Example

Suppose we declare an array of doubles as:

```
double x[] = {16.0, 12.0, 6.0, 8.0, 2.5, 12.0, 14.0, -54.5];
```

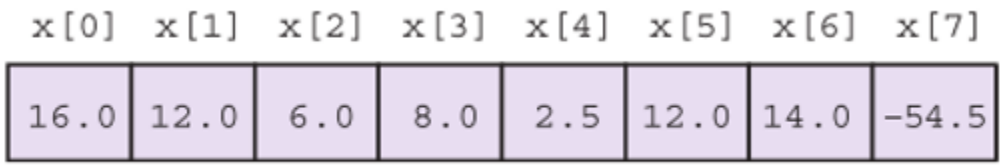


TABLE 7.2 Code Fragment That Manipulates Array x

Statement	Explanation
`i = 5;`	
`printf("%d %.1f\n", 4, x[4]);`	Displays 4 and 2.5 (value of `x[4]`)
`printf("%d %.1f\n", i, x[i]);`	Displays 5 and 12.0 (value of `x[5]`)
`printf("%.1f\n", x[i] + 1);`	Displays 13.0 (value of `x[5]` plus 1)
`printf("%.1f\n", x[i] + i);`	Displays 17.0 (value of `x[5]` plus 5)
`printf("%.1f\n", x[i + 1]);`	Displays 14.0 (value of `x[6]`)
`printf("%.1f\n", x[i + i]);`	Invalid. Attempt to display `x[10]`
`printf("%.1f\n", x[2 * i]);`	Invalid. Attempt to display `x[10]`
`printf("%.1f\n", x[2 * i - 3]);`	Displays -54.5 (value of `x[7]`)
`printf("%.1f\n", x[(int)x[4]]);`	Displays 6.0 (value of `x[2]`)
`printf("%.1f\n", x[i++]);`	Displays 12.0 (value of `x[5]`); then assigns 6 to `i`
`printf("%.1f \n", x[--i]);`	Assigns 5 (6 - 1) to `i` and then displays 12.0 (value of `x[5]`)
`x[i - 1] = x[i];`	Assigns 12.0 (value of `x[5]`) to `x[4]`
`x[i] = x[i + 1];`	Assigns 14.0 (value of `x[6]`) to `x[5]`
`x[i] - 1 = x[i];`	Illegal assignment statement

Example
Generate an array of 1000 random ints in the range 0 <= x <= 100. Calculate the average, mean, and standard deviation of the data in the array.

The standard deviation is given by:

$$standard\ deviation = \sqrt{\frac{\sum_{i=0}^{MAX\_ITEM-1} x[i]^2}{MAX\_ITEM}} - mean^2$$

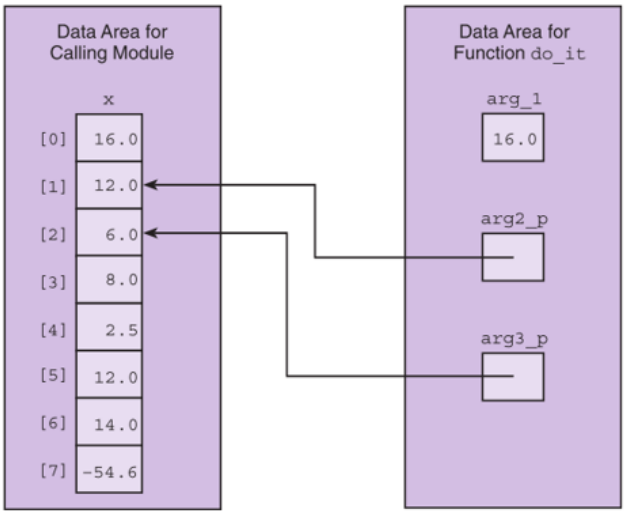
```
#define SIZE 1000
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
int main()
{
    int i;
    int data[SIZE];
    double mean, stdev, sum, sumSqr;
    srand(23);
    for(i=0;i<SIZE;i++)
       data[i] = rand() % 101;
    sum = 0;sumSqr = 0;
    for(i=0;i < SIZE;i++)
    {
        sum += data[i];
        sumSqr += data[i]*data[i];
    }
    mean = sum/SIZE;
    stdev = sqrt(sumSqr/SIZE - mean*mean);
    printf("The mean is %6.3f\n", mean);
    printf("The standard deviation is %6.3f\n", stdev);
}
```

```
The mean is 50.218
The standard deviation is 29.421
Press any key to continue . . .
```

Using Array Elements as Function Arguments

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

do_it(x[0], &x[1], &x[2]);
```



Array Arguments

- We can write functions that have arrays as arguments.
- Such functions can manipulate some, or all, of the elements corresponding to an actual array argument.

Array names are actually pointers to where the first element of an array is stored.in memory. If we index an array the index is added to the pointer to get the data.

This also means that, by default, all arrays are passed by reference and changing an array in a function that has been passed as a parameter also changes the array in the calling program.

Example
Write a function called FillArray which three parameters: the first is the array, the second is the number of items in the array, and the third is the value used to fill the array.

```
#include<stdio.h>
void FillArray(int data[], int n, int fill);
int main()
{
    int myData[20];
    int i;
    FillArray(myData, 5, 25);
    for(i=0;i<5;i++)
       printf("%d\n", myData[i]);
}
void FillArray(int data[],int n,int fill)
{
    int i;
    for(i=0;i<n;i++)
        data[i] = fill; //data and myData are the same array
}
/* printed output
25
25
25
25
25
Press any key to continue . . .
*/
```

Note that the array in FillArray is called `int data[]` but this array name is just a pointer and we could write this parameter as a pointer instead of as an array. The following also works.

```
void FillArray(int *d, int n,int fill)
{
    int i;
    for(i=0;i<n;i++)
        d[i] = fill;
}
```

In some cases, such as when multiple programmers are working on various functions that receive an array parameter you may want to pass an array but disallow the array to be changed. You can do this by declaring the array in the parameter list to be constant like this:

```
void FillArray(const int data[],int n,int fill)
{
    int i;
    for(i=0;i<n;i++)
        data[i] = fill; //this will create an error
}
```

The const declaration will prevent the array from being changed.

Example
Write a program which creates a list of 1000 simulated throws of a dice. Call a function which will count the number of ones, twos, … sixes. The function should have two parameters: The first will contain the simulation of the 1000 throws of the dice and the second will have six entries that contain the number of ones, twos, etc. The first is an input array and should be declared const, the second is an output array. The output array should be printed in the main program.

```
#define SIZE 1000
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
void CountDice(const int dice[], int numCnt[]);
int main()
{
    int dice[SIZE];
    int numCnt[7];
    int i;
    srand((unsigned)time(NULL));
    for(i=0;i < SIZE;i++)
       dice[i] = rand() % 6 + 1;
    CountDice(dice, numCnt);
    for(i=1;i<7;i++)
      printf("Number of %ds = %d\n", i, numCnt[i]);
    return 0;
}
void CountDice(const int dice[],int numCnt[])
{
    int i;
    for(i=0;i<7;i++)
        numCnt[i] = 0;
    for(i=0;i < SIZE;i++)
    {
        switch(dice[i])
        {
            case 1:
              numCnt[1]++;
              break;
            case 2:
              numCnt[2]++;
              break;
            case 3:
              numCnt[3]++;
              break;
            case 4:
              numCnt[4]++;
              break;
            case 5:
              numCnt[5]++;
              break;
            case 6:
              numCnt[6]++;
              break;
            default:
              break;
        }
    }
}
```

Note that in general, for case I we increment numCnt[i]. This allows us to write it like this:

```
void CountDice(const int dice[],int numCnt[])
{
    int i;
    for(i=0;i<7;i++)
        numCnt[i] = 0;
    for(i=0;i<SIZE;i++)
        numCnt[dice[i]]++;

}
```

The statement `numCnt[dice[i]]++` is calculating the name of a variable.

> You may be tempted to write a function which returns an array. In C functions may not return an array so they must be used as output parameters.

Example
Write a function which will accept and array as a parameter and return the index of the first negative entry in the array. If no negative entries are found, your function should return -1.

```
#include<stdio.h>
int FirstNegative(int d[], int n);
int main()
{
    int d[] = {1, 2, 3, 4, -1, -2, -3, -4};
    int indxNeg;
    indxNeg = FirstNegative(d, 8);
    printf("Index of first negative is %d\n", indxNeg);
}
int FirstNegative(int d[], int n)
{
    int indx = 0;
    while(indx < n && d[indx] >= 0)
        indx++;
    if(indx == n)
       indx = -1;
    return indx;
}
```

CS 210 **October 4, 2016**
Arrays

Write a method called Rotate which shifts all the values in an array argument to the left 1 place with the value at index 0 becoming the last value in the array. For example, if the array is defined by `int [] data = {4, 5, 6, 7}`, your method should return the array with the data `{5, 6, 7, 4}`.

Use the following sequence in the main program to call your function.

```
#include<stdio.h>
void Rotate(int d[], int n);
int main()
{
    int d[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    int i;
    for(i=0;i<8;i++)
       printf("%d ", d[i]);
    printf("\n");
    Rotate(d, 8);
    for(i=0;i<8;i++)
       printf("%d ", d[i]);
    printf("\n");
}
```

Turn in a printed copy of your source file.

Press any key to continue . . .