Lab-7: Array

An *array* is a collection of data storage locations, each having the same data type and the same name. Each storage location in an array is called an *array element*. Imagine that you're designing a program to keep track of your business expenses. The program could declare 12 separate variables, one for each month's expense total. This approach is analogous to having 12 separate folders for your receipts. Good programming practice, however, would utilize an array with 12 elements, storing each month's total in the corresponding array element. This approach is comparable to filing your receipts in a single folder with 12 compartments.

Single-Dimensional Arrays

A *single-dimensional array* has only a single subscript. A *subscript* is a number in brackets that follows an array's name. This number can identify the number of individual elements in the array. An example should make this clear. For the business expenses program, you could use the following line to declare an array of type float:

```
float expenses[12];
```

The array is named expenses, and it contains 12 elements. Each of the 12 elements is the exact equivalent of a single float variable. All of C's data types can be used for arrays. C array elements are always numbered starting at 0, so the 12 elements of expenses are numbered 0 through 11. In the preceding example, January's expense total would be stored in expenses[0], February's in expenses[1], and so on. When you declare an array, the compiler sets aside a block of memory large enough to hold the entire array. Individual array elements are stored in sequential memory locations.

The location of array declarations in your source code is important. As with nonarray variables, the declaration's location affects how your program can use the array. For now, place your array declarations with other variable declarations, just in the beginning of main().

Exercise 1: Type the following code and match your output with the given output.

```
#include <stdio.h>

main()
{
   float expenses[13];
   int count;

    for (count = 1; count < 13; count++)
    {
        printf("Enter expenses for month %d: ", count);
        scanf("%f", &expenses[count]);
    }


    for (count = 1; count < 13; count++)
    {
        printf("Month %d = $%.2f\n", count, expenses[count]);
    }
    return 0;
}
```

Output

```
Enter expenses for month 1: 100
Enter expenses for month 2: 200.12
Enter expenses for month 3: 150.50
Enter expenses for month 4: 300
Enter expenses for month 5: 100.50
Enter expenses for month 6: 34.25
Enter expenses for month 7: 45.75
Enter expenses for month 8: 195.00
Enter expenses for month 9: 123.45
Enter expenses for month 10: 111.11
Enter expenses for month 11: 222.20
Enter expenses for month 12: 120.00
Month 1 = $100.00
Month 2 = $200.12
Month 3 = $150.50
Month 4 = $300.00
Month 5 = $100.50
Month 6 = $34.25
Month 7 = $45.75
Month 8 = $195.00
Month 9 = $123.45
Month 10 = $111.11
Month 11 = $222.20
Month 12 = $120.00
```

Exercise 2: In this programme have to average the score of a 10 person individual score using a single array and a single for loop. Your output should be as follows:

```
Output:
Enter Person 1's grade: 95
Enter Person 2's grade: 100
Enter Person 3's grade: 60
Enter Person 4's grade: 105
The highest grade possible is 100
Enter correct grade: 100
Enter Person 5's grade: 25
Enter Person 6's grade: 0
Enter Person 7's grade: 85
Enter Person 8's grade: 85
Enter Person 9's grade: 95
Enter Person 10's grade: 85
The average score is 73
```

Exercise 3: Follow the following steps:

1. The program in this here declares and initializes an array a in which each element has the same value as its subscript. i.e. a[i] = i ;
2. Add another loop that will multiply all the even numbers in the array by
3. (Recall that to check if a number is even you need to use code of the form if (num%2==0) .....).
4. Add a third loop to display all the values in the array on one line with a single space between each value Compile and run the program. The output should be

4. Output will be : 0 1 6 3 12 5 18 7 24 9

Exercise 4 : Fibonacci Sequence:

1. Declare an array of 100 int without initialising the elements
2. use two assignment statements to set the first two elements to be equal to 0 and 1 respecti6vely.
3. use a loop to set the values of the remaining elements, so that (apart from the first two elements) each element in the array is equal to the sum of the previous two elements in the array
4. Compile and run the program.
5. if you input 15

The output should be

The 15^{th} series is: 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377

Exercise 5: Searching minimum element. So, what is this "minimum element " algorithm? The basic idea is that the smallest element in the array will be the 0^{th} element in the array. Let array be the array of integers, and num_elements be the total number of elements in array.

1. Start with start = 0 (for the index of the zeroth element).
2. Set smallest = start (smallest stores the index of the smallest element encountered so far).
3. Run through a loop with the variable index going from start to num_elements o If array[index] < array[smallest], set smallest = index.
4. Once the loop ends, swap array[start] and array[smallest](moving the smallest element found to the beginning of the array you searched).
5. We are done array[start] the minimum number, just return it.

Now find the maximum num of that Array. it is quite easy Right ? Ok Do Now Task 1, 2 and 3 using multidimensional array.

Home Work:

1. The minimum element sort algorithm: The basic idea is that the smallest element in the array will be the 0^{th} element in the sorted array, the second-smallest will be the first element, etc. Here's how it works: Let array be the array of integers, and num_elements be the total number of elements in array.

- Start with start = 0 (for the index of the zeroth element).
- Set smallest = start (smallest stores the index of the smallest element encountered so far).
- Run through a loop with the variable index going from start to num_elements
- 4. If array[index] < array[smallest], set smallest = index.
- Once the loop ends, swap array[start] and array[smallest](moving the smallest element ofound to the beginning of the array you searched).
- Increment start, and if start < num_elements, go back to step 2. Don't use a goto statement, rather, use another loop.
- If start >= num_elements then you're done and the array is sorted.

2. Multiplication of 2, 3*3 matrix.

Thanks

Md. Shamsujjoha
Faculty member, Department of Computer Science & Engineering, East West University
Email: dishacse@yahoo.com