CL1002 P*rogramming* ***Fundamentals Lab***	Lab 7 Introduction to Arrays, Loops with Arrays

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

Fall 2024

AIMS AND OBJECTIVES

Aims:

1. **Automating Repetitive Tasks:**
   - To eliminate manual repetition by automating repetitive tasks, making the code more efficient and easier to manage.
2. **Efficient Code Execution:**
   - To enhance the efficiency of programs by reducing the amount of code required for repeating operations, leading to faster execution.
3. **Improve Code Readability and Maintainability:**
   - To improve the readability and maintainability of the code by using concise structures to handle repetitive processes rather than long, duplicated blocks of code.
4. **Control Flow Management:**
   - To give programmers more control over the flow of a program, allowing loops to repeatedly execute a block of code based on specified conditions.
5. **Dynamic Input Handling:**
   - To process dynamic inputs or continuously check conditions, allowing for flexible and adaptive behavior of programs, such as waiting for user input or processing a sequence of data elements.

Objectives:

1. **Iterate Over Data Structures:**
   - To allow for iteration over data structures like arrays, linked lists, or any other collections without writing repetitive code for each element.
2. **Terminate on Specific Conditions:**
   - To ensure that loops execute only as long as a particular condition is met, allowing the program to halt the loop once the desired outcome is achieved.
3. **Reduce Code Complexity:**
   - To simplify complex operations by reducing the number of lines of code and providing a clean and logical way to repeat tasks.
4. **Support Event-Driven Programming:**
   - To enable continuous event-checking mechanisms in real-time applications where the program needs to monitor inputs, actions, or other triggers without terminating.
5. **Flexible Program Execution:**
   - To offer flexible programming control by allowing loops to start, continue, or terminate at any point based on the evaluation of conditions (e.g., using break, continue statements).

ARRAYS IN C

An array in C is a collection of elements of the same data type stored in contiguous memory locations. A single-dimensional array, often referred to as a linear array, is the simplest form of an array. It is essentially a list of elements that can be accessed by a single index.

Declaring a Single-Dimensional Array in C

To declare a single-dimensional array in C, you use the following syntax:

```
01 data_type array_name[array_size]; //Uninitialized
02 data_type array_name[array_size]={initialization list}
```

Where:

- data_type is the type of data the array will store (e.g., int, float, char).
- array_name is the name of the array.
- array_size is the number of elements in the array.

INTERPRETATION

The general uninitialized array declaration allocates storage space for array aname consisting of size memory cells.

● Each memory cell can store one data item whose data type is specified by element-type (i.e., double, int , or char).

● The individual array elements are referenced by the subscripted variables aname [0] , aname [1] , . . , aname [size −1] .

● A constant expression of type int is used to specify an array's size . In the initialized array declaration shown, the size shown in brackets is optional since the array's size can also be indicated by the length of the initialization list.

● The initialization list consists of constant expressions of the appropriate element-type separated by commas.

● Element 0 of the array being initialized is set to the first entry in the initialization list , element 1 to the second and so forth.

MEMORY REPRESENTATION

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

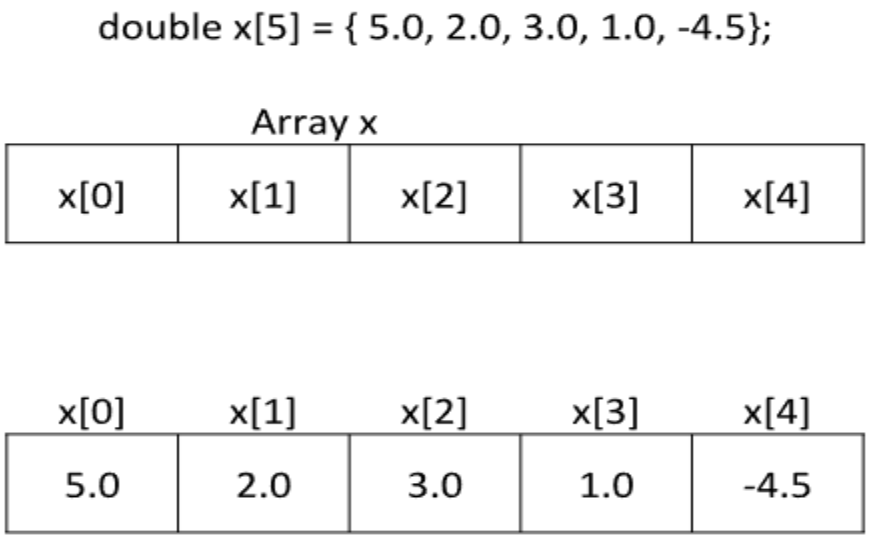

Figure 1. Memory Representation of an Array

EXAMPLE 1

```c
#include <stdio.h>

   int main() {
       // Declaring an array of integers with 5 elements
       int numbers[5] = {10, 20, 30, 40, 50};

       // Accessing array elements
       for (int i = 0; i < 5; i++) {
           printf("numbers[%d] = %d\n", i, numbers[i]);
       }

       return 0;
   }
```

Key Points:

- **Indexing**: The index of the array starts from 0. For example, numbers[0] refers to the first element, and numbers[4] refers to the last element in an array of 5 elements.
- **Initialization**: You can initialize an array at the time of declaration, as shown in the example. If not, the array elements will contain garbage values unless explicitly initialized.
- **Accessing Elements**: Array elements are accessed using the index, as shown in the printf statement of the example.

EXAMPLE 2

Array length: You need to specify the size of the array at compile time, or you can calculate it like this:

```
01 int size = sizeof(array_name) / sizeof(array_name[0]);
```

This is useful if you want to traverse a user defined array but do not know the actual size of the array.

Modifying elements: You can assign new values to specific elements by referring to their index:

```
01 numbers[2] = 60;  // Changes the 3rd element to 60
```

Here we are changing the index 2 value which the 3rd element in the array from 30 to 60

CHARACTER ARRAY

A character array in C is an array that stores a sequence of characters. It is commonly used for working with strings, which are essentially arrays of characters ending with a null terminator (`'\0'`).

Declaring and Initializing a Character Array

A character array can be declared and initialized in multiple ways. Here are some common methods:

Declaration without Initialization:

```
01 char str[10]; // Array to store up to 9 characters,
   leaving space for the null terminator
```

Declaration with Initialization:

You can initialize a character array in two ways:

a) Using individual characters:

```
02    char str[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

Here, the last element is the null terminator '\0' to indicate the end of the string.

b) Using a string literal:

```
03    char str[6] = "Hello";
```

C automatically adds the null terminator at the end of the string.

ACCESSING AND MODIFYING CHARACTER ARRAYS

Accessing Elements

You can access individual characters in a character array using indexing, just like with any other array.

```
04    #include <stdio.h>
05
06    int main() {
07        char str[] = "Hello";
08
09        // Accessing and printing each character
10        for (int i = 0; i < 5; i++) {
11            printf("Character at index %d: %c\n", i,
str[i]);
12        }
13          str[0]='J';
14        for (int i = 0; i < 5; i++) {
15            printf("Character at index %d: %c\n", i,
str[i]);
16        }
17
18
19        return 0;
20    }
```

Here I have traversed the array as well as modified the first index to make it print “HELLO” then “JELLO”

SCANSET

scanf family functions support scanset specifiers which are represented by %[]. Inside scanset, we can specify single character or range of characters. While processing scanset, scanf will process only those characters which are part of scanset. This is useful when you want to match a specific range or set of characters from user input.

EXAMPLE 3

Here is an example of how a scanset works with scanf():

```
01 #include <stdio.h>
02
03 int main() {
04     char str[100];
05
06     // Scanset example: accepts only alphabetic
   characters (A-Z, a-z)
07     printf("Enter a string: ");
08     scanf("%[A-Za-z]", str);
09
10     printf("You entered: %s\n", str);
11     return 0;
12 }
```

%[A-Za-z] is a scanset. It tells scanf() to read and store characters that match the set of letters from A to Z (both lowercase and uppercase).

The input reading will stop when a character outside this range (non-alphabetic) is encountered.

NEGATING A SCANSET

You can also negate a scanset using the ^ symbol. This means "accept everything except the characters specified."

```
01 #include <stdio.h>
02
03 int main() {
04     char str[100];
05
06     // Scanset example: accepts everything except
   alphabetic characters
07     printf("Enter a string: ");
08     scanf("%^[A-Za-z]", str);
09
10     printf("You entered: %s\n", str);
11     return 0;
12 }
```

Here, %[^A-Za-z] tells scanf() to read and store any character *except* those between A and Z (upper and lowercase).

Problems:

1. Create a program that takes an array of size 6 and shifts all its elements to the right by one position. The last element should move to the first position.
   Input: {1,2,3,4,5,6}
   Output: {6,1,2,3,4,5}

2. Write a program that takes a string as input from the user and counts the frequency of each vowel (A, E, I, O, U) in the string
3. Input: Hello World
   Output: a=0, e=1, I=0, o=2, u= 0

4. Write a program that calculates the length of a string using a pointer. Do not use the built-in strlen() or sizeof() function. After length calculation reverse the string using the same pointers

5. Write a program which identifies if a given string is a palindrome using arrays.

6. Write a program to read 10 integers into an array. Then, use loops to find the frequency of each element in the array (how many times each number occurs).

7. Given an array arr[] of size N which contains elements from 0 to N-1, you need to find one of the elements occurring more than once in the given array.
   Input:
   Array Size =5
   Element 1=1
   Element 2=2
   Element 3=3
   Element 4=2
   Element 5=5
   Output:
   Number 2 occur more than once.

   Note: You cannot utilize nested loops.

8. You are required to design a calculator utilizing an array. Take a string from the user i.e. 20+10-30 and store it in an array. Traverse through the array, if there are values stored them in some variable and if an operation comes perform the necessary operation and if a null character comes display the result. If the null character comes after the operator, the program should print an invalid expression. (Note: Only do this for + and - operator)