M03S01 [Arrays and Strings]: Practice Drill

CS110: Computing Lab
Department of CSE, IIT Guwahati
Jan-May 2018

Objective of the Stage

Primary objective of this drill is to make you practice with arrays. In the first stage of the Module 3, we have focused on one dimensional arrays. In the subsequent stages, practice and assessment problems will have problems on multi-dimensional arrays. Strings in C are represented by arrays of characters.

Reference Lecture Slides

- 24th Jan Lecture [Alternate class: Deepanjan Kesh Slide] [`https://goo.gl/NFmpr5`]
- 1st Feb Lecture [Array and Pointer I: May ignore Pointer discussion for this practice drill] [`https://goo.gl/y82yzt`]

Learning Goal

After doing this practice drill and working on assessment problems of this stage, we expect you to be comfortable with

- basic understanding of array
- declaration, initialization of a one-dimensional array
- accessing individual array element and performing some basic operations
- applying a sorting technique on data stored in an array
- strings and storing them in arrays,
- performing some basic operations with strings

Arrays

All the variables used so far had a common characteristic: each variable can only be used to store a single value at a time. For example, variable *marks* in `float marks` can only store marks of a student. But defining a hundreds of such variables for a class of 100 students does not seem a good idea. An **array** offers a convenient way to group together related variables in one or more dimensions.

An **array** is a collection of variables of the *same type* that are referred to by a *common name*.

For example, we can store marks of 100 students in a `float` type array `marks`, written as `float marks[100]`.

One-Dimensional Arrays

A one-dimensional array is a list of related variables. The general form of a one-dimensional array declaration is:

type variable_name[size],

where type is base type of the array (e.g. `int`) determining the data type of each element in the array; size indicates the number of elements the array can contain; and *variable_name* is the name of the array.

The size of an array must be an `int` constant or a constant expression.

Examples:
`int count[10]` indicates *count* is an array of type `int` which can hold 10 integers;
`float marks[100]` indicates *marks* is an array of type `float` which can hold 100 numbers;
`char name[40]` indicates *name* is an array of type `character` which can hold 40 characters;
`float marks[n]` indicates *marks* is an array of type `float` which can hold n numbers, where n is an `int` variable with constant value;

Array Initialization

Arrays can be initialized during declaration as shown below:

- `int A[10] = {1,4,5,2,3,7,6,8,9,10}`
- `double A[ ] = {1.2,3.4,6.7}`

In the second example, the C compiler will deduce the size of the array from the initialization statement.

C does not allow declaring an array whose number of elements is unknown at compile time. So the following declaration is invalid:

```
double A[ ]
```

Now consider the following code:

```
#include <stdio.h>
int main()
{
   int n;
    float x[n];
    ...
    scanf("%d",&n);
    return 0;
}
```

Here, the variable size array declaration is used, where n is an integer variable. This will result in a compile time error.

Sometimes it is convenient to define an array size in terms of a symbolic constant, rather than a fixed integer quantity. So, writing the above code as follows will not lead to any error due to array declaration:

```
#include <stdio.h>
#define n 10
int main()
{
    float x[n];
    ...

    return 0;
}
```

Array can be initialized by sequentially assigning a value to each array element. The following example initialize an integer array `A` using a `for` loop.

```
#include <stdio.h>

int main()
{

    int A[10];
   for(i = 0; i < 10; i i++)
    {
      A[i] = i+1;
    }

}
```

Array Accessing

In the last example of the previous section, we saw that array elements are referred to by their position in the list. `A[0]` refers to the first member of the array A, `A[1]` refers to the

2nd member and so on. i^{th} member of the array A can be accessed by `A[i-1]`.

Basically, the array index starts at 0 and ends at *size* - 1.

Important thing to remember

WHile working with arrays in C, there are few important things to remember as -

1. C does not check for bounds. This means even if you assign values to an array element which is beyond the `size` of array, compiler will not produce an error.

```c
/* An incorrect C code for which C Compiler will not show error, but it may
   give error during runtime. */
int main()
{
int test[10], i;

for(i=0;i<20;i++) test[i] = i;

return 0;
}
```

2. C does not allow a direct assignment of one array to another. So the following code fragment is incorrect:

```c
int a[10], b[10];
a = b; /* Trying to assign all elements of array b to array a*/
```

Instead, it has to be done for each element in the following way:

```c
int a[10], b[10];
for(i=0;i<10;i++)
   a[i] = b[i]; /* Trying to assign all elements of array b to array a*/
```

Practice Exercises on one-dimensional arrays

Now, we ask you to do some practice problems given below. This time we are not providing you the code for these problems but ask you to try first by yourself and if you face difficulty, take help of any textbook or TA during the lab session.

1. Write a program in C to store elements in an array and print it on the screen. Your program should read the inputs from user during runtime.

2. Write a program in C to count a total number of duplicate elements in an array. Use any random number generator to generate integers between 1 and 10, and use these numbers to initialize array.

3. Write a program in C to find the maximum, minimum, 2nd largest and 2nd smallest element in an array.

4. Write a program in C to sort elements of the array in descending order.

Strings

In C, a **string** is defined as a character array terminated by a special character, the *null character* represented by `\0`. The example below show a array representation of the string *Hi*.

‘H’	‘i’	‘ “0’

STRING DECLARATION: As discussed above string is an array of characters, hence the declaration of a 10-character string will look like:

```
   char str[11]
```

Carefully check the size of the string during declaration. You have to make sure that the special *null character* has been included in your character count.

STRING INITIALIZATION: Character arrays or strings can also be initialized as follows:

```
   char str[9] = ‘‘I like C’’;
```

which is the same as

```
   char str[9] = {‘I’,‘ ’,‘l’,‘i’,‘k’,‘e’,‘ ’,‘C’,‘\0’}
```

In other words, whenever a string is enclosed in double quotes, C automatically creates an array of characters containing that string and terminated by the *null character*. So C allows us to write:

```
   char str[ ] = ‘‘Hi’’;
```

But it will generate an error if we write

```
   char str[2] = ‘‘Hi’’;
```

FIND OUT WHY?

Important Functions

Below is the list of functions which are important while working with strings or arrays of characters. Please go through with each of the function using the text-book.

1. Functions for String Input: `scanf` with %s; `gets`

2. Printing Strings: `printf` with %s; `puts`

3. Familiarize yourself with *character functions* in `ctype.h`. Few examples `ialnum`, `isalpha`, `islower`, `tolower` etc.

4. Familiarize yourself with string manipulation functions available with `string.h` library. Few examples are `strcpy`, `strcat`, `strlen`, `strcmp`, `strchr`, `strstr`.

Practice Problems

Now is the time for some practice problems.

1. Write a program in C to input a string and print it.
2. Write a program in C to count the total number of words and characters in a string.
3. Write a program in C to extract a substring from a given string
4. Write a program in C to read a sentence and replace lowercase characters by uppercase and vice-versa. First use appropriate string library function and then write another program which does not use any string library function.