CMSC 11: ARRAYS & PROCESSING OF ARRAYS

by: Maverick Crisostomo

Objectives:

At the end of the lesson you should:

1. Have learned the basics of using arrays;
2. Be able to define a type using typedef; and
3. Have learned the concepts of program correctness.

ARRAYS

Arrays are used to represent structures of consecutive elements of the same type. Let's look at the syntax for declaring an array.

Syntax:

<data-type> <arrayName>[<array-size>];

Examples:

int intArray[100]; //an array that can contain 100 integers.

float floatGrades[4]; //an array that can contain 4 floats.

char charArray[50]; //an array of characters or a *String*.

Example Code:

```
#include <stdio.h>

int main()
{
        int intArray[3];  // can hold 3 items

        /* initialize the array items */
        intArray[0] = 5;
        intArray[1] = -2;
        intArray[2] = 23;

        /* output... */
        printf("The first item in the array is %d\n", intArray[0]);
        printf("The last item in the array is %d\n", intArray[2]);

}
```

Explanations:

- *Array declaration*
    - Start with the ***type***. Then, ***name*** for the array. Followed by brackets (***[ ]***) containing an ***integer*** that specifies the number of items that can be stored into the array.
    - ***type name[dim];***
- *Array initialization.*
    - *Declaring* an array does not *initialize* its values. Your code will need to set a value for each item in the array.
- *Accessing an item in the array to set or read its values.*
    - Use variable name followed by brackets containing the ***index*** of the desired item.
    - ***Index*** values **start at 0,** not 1. Valid index values are from 0 to n-1, where n is the size of the array.

USING LOOPS TO ACCESS ARRAYS

Now try modifying the sample code above such that the initial values of the array will be taken from the user's input using loops. And try printing the contents of the array. Your code should be like this:

```
#include <stdio.h>

main(){
        int intArray[3];  // can hold 3 items
        int i;

        /* initialize the array items */
        for (i = 0; i < 3; i++)
        {
                printf("Please enter a value");
                printf(" for item %d: ", i+1);
                scanf("%d", &intArray[i]);
        }

        /* output... */
        for (i = 0; i < 3; i++)
                printf("Value = %d\n", intArray[i]);

}
```

STRINGS AS ARRAYS OF CHARACTERS

A string is a series of "char" type data terminated by a NULL character, which is '/0'.

```
#include <stdio.h>

main( )
{
      char name[5]; /* define a string of characters */

      name[0] = 'D';
      name[1] = 'a';
      name[2] = 'v';
      name[3] = 'e';
      name[4] = '\0'; /* Null character - end of text */

      printf("The name is %s\n",name);
      printf("One letter is %c\n",name[2]);
      printf("Part of the name is %s\n",&name[1]);
}
```

The variable '***name'*** is therefore a string which can hold up to 4 characters and one ***null*** character at the end to terminate the string.

SOME STRING SUBROUTINES

```
1   #include <stdio.h>
2
3   main( )
4   {
5      char name1[12],name2[12],mixed[25];
6      char title[20];
7
8      printf("enter your name: ");
9      scanf("%s", name1);
10
11     printf("enter another name: ");
12     fflush(stdin);  /* use this to clear the buffer */
13     gets(name2);
14
15     strcpy(title,"This is the title.");
16
17     printf(" %s\n\n",title);
18     printf("Name 1 is %s\n",name1);
19     printf("Name 2 is %s\n",name2);
20
21     if(strcmp(name1,name2)>0) /* return 1 if name1 > name2; -1 if name1 <
22     name2 */
23        strcpy(mixed,name1);
24     else
25        strcpy(mixed,name2);
26
```

```
27      printf("The biggest name alphabetically is %s\n",mixed);
28
29      strcpy(mixed,name1);
30      strcat(mixed," ");
31      strcat(mixed,name2);
32
33      printf("Both names are %s\n",mixed);
    }
```

Things to notice…

- ***scanf()*** and ***gets()*** (*see Lines 9 and 13*) are functions for getting input from user.
- ***strcpy(s,ct)*** -> copy ct into s, including "\0"; return s. (*see Lines 15, 22, 24 and 28*)
- ***strcmp(cs,ct)*** -> compare cs and ct; return 0 if cs=ct (*see Line 21*).
- ***strcat(s,ct) ->*** concatenate ct to end of s; return s. (*see Lines 29 and 30*).
- ***strlen(cs)*** -> return length of cs

TYPEDEF

typedef declarations are made to construct shorter or more meaningful names for types already defined by C or for types that you have declared. A typedef declaration is interpreted in the same way as a variable or function declaration, but the identifier, instead of assuming the type specified by the declaration, becomes a synonym for the type.

Syntax:

typedef type-definition identifier;

The statement above assigns the symbol name *identifier* to the data type definition *type-definition*. For example,

```
typedef char byte;
typedef char string[41];
typedef struct {float re, im;} complex;
typedef char *byteptr;
```

After these definition, you can declare

```
byte m, n;
string myStr;
complex z1, z2;
byteptr p;
```

User defined types may be used at any place where the built-in types may be used.

```
char m, n;
char myStr[41];
struct {float re, im;} z1, z2;
char *p;
```

PRACTICE EXERCISES:

1. Declare one integer array, with 10 elements, called “array1”. Using a loop, put some data in each and get the average, the min value and the max value.
2. Write a program with three short strings, about 6 characters each, and use “strcpy” to copy “one”, “two”, and “three” into them. Concatenate the three strings (without using the ‘strcat()’ function) into one character array named “strings”.
3. Create a data type named “string” which is a character array of length 25. Declare two variables of the type you have created. Read two input strings from the user and store it in the variable you have declared, and print it in reverse order.