C-Programming Practice (Part – 2)

# 1. 1-D Arrays in C with functions and Strings

Arrays are collections of elements all of the same type. The elements go into memory, one after the other. If an array is declared as: **int values[5];**

then the array **values** has 5 elements; they are values[0], values[1], values[2], values[3], and values[4]. Notice that the index (subscript) starts from ***0*** to ***ArraySize - 1***. The elements of the array behave like normal variables. In order to use any element of the array, we need to use the array name plus the index value. You can initialize an array when you declare it using the initialization list:

**int scores[5] = {10, 20, -5, 40, 5};**

Consider the following initialization statement: **int x[6]= {2, -6, 7};** the initialization will create an array of 6 elements; the first 3 elements will be assigned the values 2, -6, and 7 in order i.e. x[0]=2, x[1]=-6, and x[2]=7. Each of the remaining 3 elements of the array which are x[3], x[4], and x[5] will be assigned a value of 0.

The following declaration will create a compilation error because of not specifying the array size:

**double x[]; // ERROR IF NOT A FORMAL PARAMETER DECLARATION**

but the following declaration and initialization

**double x[] ={3,7,8,11};**

is accepted because in this case the compiler will count the number of elements in the list and make the array size equal to that number which is 4 in the above example.

Example: Use of Arrays with a function

**#include <stdio.h>**

**#define SIZE 10**

**int sumValues (int a[], int n ); //function prototype**

**int main() {**

**int values[SIZE];**

**int total\_sum,i;**

**printf("Enter %d integer values> ",SIZE);**

**for(i=0; i < SIZE; i++)**

**scanf("%d",&values[i]);**

**total\_sum = sumValues (values, SIZE); //function call**

**//The function accesses directly the elements of array values.**

**printf("Total sum=%d\n", total\_sum);**

**return 0;**

**}**

**// function definition**

**int sumValues (int a[], int n) {**

**int i, sum = 0;**

**for(i = 0; i < n; i++)**

**sum += a[i];**

**return sum;**

**}**

**Exercise 1**

#### **Write a C program that declares 3 integer 1-D arrays x, y, and z of size SIZE (constant to be defined equal to 10). The program then reads values for x and y arrays from the user. For reading, create a function read\_array. (void read\_array(int array[], int n) )**

#### **Finally the program finds the common values in x and y. To achieve this, you need to write a function intersection that receives 2 integer arrays as input arguments and a third array z to be used as output to contain the common values. In addition to putting the common values in array z, the function returns the number of common values. After calling the intersection function, your main function will display the contents of the three arrays on the screen. For printing, create and use the function print\_array. (void print\_array(int array[], int n) )**

**Strings in C**

* A string is a list of characters enclosed in double quotes. It is represented in C as an array of characters. However, one additional cell is required for storing the terminating character, **‘\0’**,called **NULL**

Example: Even though the string ”I like C” has 8 characters, we need to declare an array of size 9. **char str[9]=”I like C”;** Note that in the example above, the null character is automatically inserted by the compiler

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **‘I’** | **‘ ‘** | **‘l’** | **‘i’** | **‘k’** | **‘e’** | **‘ ‘** | **‘C’** | **‘\0’** |

* All the string functions in C require the presence of the **NULL** character to work correctly.
* Strings can also be declared without specifying the size as in: **char str[]=”I like C”;**

In this case, sufficient storage is allocated including that for **NULL.**

* Like array, the individual characters of a string can be accessed by specifying the index. Example, assuming the above assignment, **str[4]** is the character **‘k’**

**String I/O**

* Use the **fgets** function to read a string. fgets allows you to read a string that could include blanks. It terminates reading on encountering the return character. However, the return character will be part of the string. Also the easiest way to output string is by using the **puts**. function as in: puts(name); [
* You can also use the scanf/printf functions for string I/O using the %s format specifier. However, scanf terminates reading on encountering the blank character, so first and last names must be read separately:

Example

char first[10],last[10];

printf(”\nEnter your first name: ”);

scanf(“%s”,first);

printf(”\nEnter your last name: ”);

scanf(“%s”,last);

printf(“\nYour full name is : %s %s”,first,last);

* Note that the address of operator, &, is not used in the scanf function.

Built-in string functions:

Built-in string processing functions can be accessed through the header file, string.h . Some are:

* strcat(string1, string2) appends string2 to string1
* strcpy(string1, string2) replaces string1 with string2
* strlen(string1) returns the length of string1 excluding NULL character
* strchr(string, ch) searches for the occurrence of ch in string
* strstr(string1, string2) searches for the occurrence of string2 in string1
* strcmp(string1, string2) compares string1 and string2 and returns:

Negative integer if string1 < string2

0 if string1 = = string2

positive integer if string1 > string2.

* The following program illustrates the use of some string functions:

#include <string.h>

#include <stdio.h>

int main() {

char s1[81], s2[81];

fgets(s1, 80, stdin); //Try gets here

fgets(s2, 80, stdin); //Try gets here

printf("lengths: %d %d\n", strlen(s1), strlen(s2));

if (!strcmp(s1, s2))

printf("The strings are equal\n");

strcat(s1, s2);

printf("%s\n",s1);

return 0;

}

* Note: It is important to remember that strcmp() returns false if the strings are equal, so be sure to use ! to reverse the condition

Exercise 2

**Write a function search\_char that receives a string and a character and returns the index of the character if it is present in the string or -1 otherwise. The function is similar to linear search function but works with characters. Write a main function to test your function. Prompt the user for input.**

Exercise 3

**Write a function ispalindrome that receives a string returns back as int (0 or 1) if the string is a palindrome or not. Write a main function to test your function. Prompt the user for input.**

# 2. 2-D Arrays in C with functions

Example and Exercise 4:

**Study the following program, compile and execute it. Then provide the code for the function add\_2d\_arrays by putting the result of adding matrices a and b into c. Check your output.**

#include<stdio.h>

#define ROWS 4

#define COLS 4

void read\_2d\_array(int [][COLS], int rows, int cols);

void add\_2d\_arrays(int a[][COLS], int b[][COLS], int c[][COLS], int rows, int cols);

void print\_2d\_array(int a[][COLS], int rows, int cols);

int main(void) {

int i, j, a[ROWS][COLS], b[ROWS][COLS], c[ROWS][COLS];

int rows, cols;

printf("Enter number of rows for the Matrix: ");

scanf("%d", &rows);

printf("Enter number of columns for the Matrix: ");

scanf("%d", &cols);

read\_2d\_array(a, rows, cols); // reading matrix a

read\_2d\_array(b, rows, cols); // reading matrix b

add\_2d\_arrays(a, b, c, rows, cols); /\* Addition of two matrices \*/

printf("The sum of two matrices is: \n");

print\_2d\_array(c, rows, cols); /\*Print sum of two matrices \*/

return 0;

}

void read\_2d\_array(int a[][COLS], int rows, int cols) {

int i, j;

printf("Enter the %d elements of the 2-D array row-wise: \n", rows \* cols);

for(i=0; i<rows; i++) {

for(j=0; j<cols; j++)

scanf("%d", &a[i][j]);

}

}

void add\_2d\_arrays(int a[][COLS], int b[][COLS], int c[][COLS], int rows, int cols){}

void print\_2d\_array(int a[][COLS], int rows, int cols) {

int i, j;

for(i=0; i<rows; i++) {

for (j=0; j<cols; j++)

printf("%5d ", a[i][j]);

printf("\n");

}

}

Exercise 5

**Write a program in C that reads an 2-Dimensional array from the user of size ROW x COL. It then finds and prints the transpose of the array.**

# 4. Structures

Similar to classes in Java as in the following example

#include <stdio.h>

**struct Distance{**

**int feet;**

**float inch;**

**} d1, d2, sum;**

int main(){

printf("Enter information for 1st distance\n");

printf("Enter feet: ");

scanf("%d", &**d1.feet**);

printf("Enter inch: ");

scanf("%f", **&d1.inch**);

printf("\nEnter information for 2nd distance\n");

printf("Enter feet: ");

scanf("%d", **&d2.feet**);

printf("Enter inch: ");

scanf("%f", **&d2.inch**);

sum.feet = d1.feet + d2.feet;

sum.inch = d1.inch + d2.inch;

/\* If inch is greater than 12, changing it to feet. \*/

if (sum.inch>12.0)

{

sum.inch=sum.inch-12.0;

++sum.feet;

}

printf("\nSum of distances=%d\'-%.1f\"\n",sum.feet,sum.inch);

return 0;

}

**Exercise 6**

**Write a structure called Date that has three fields (int day, int month and int year). For this exercise, assume that February (month 2) has 28 days (so you ignore leap years).**

**Now write a main function that accepts from the user the three values of day, month and year.**

**Using the structure as defined above write a function that returns the date after 7 days. Write a main program with your output should be as follows:**

**Enter the current date (day month year)> 30 08 2020**

**After 7 days the new date will be 6 September 2020.**