Chapter 7

Arrays and Strings

Arrays

An Array is a fixed-size sequenced collection of elements of the same data type. It is simply a grouping of like-type data.

Suppose we had a set of grades that we wished to read into the computer and suppose we wished to perform some operations on these grades, we will quickly realize that we cannot perform such an operation until each and every grade has been entered since it would be quite a tedious task to declare each and every student grade as a variable especially since there may be a very large number. In C we can define variable called grades, which represents not a single value of grade but entire set of grades. Each element of the set can then be referenced by means of a number called as index number or subscript.

One-Dimensional Arrays

A list of items can be given one variable name using only one subscript and such a variable is called a *single-subscripted variable* or a *one-dimensional* array.

Declaration of One-Dimensional Arrays

Like any other variable arrays must be declared before they are used. The general form of declaration is: ***type variable_name[50];***

The *type* specifies the type of the elements that will be contained in the array, such as int, float or char and the size indicates the maximum number of elements that can be stored inside the array. For example: **float height[50];** Declares the height to be an array containing 50 real elements. Any subscripts 0 to 49 are valid. In C the array elements index or subscript begins with number zero. So height [0] refers to the first element of the array. (For this reason, it is easier to think of it as referring to element number zero, rather than as referring to the first element).

Global Declaration

```
char string[50]; /* Global Declaration */

void main(){

        …………………

}
```

Local Declaration

```
void main(){

        char age[] = {21,22,19,20}; /* Local Declaration & Initialization */

        …………………

}
```

As individual array element can be used anywhere that a normal variable with a statement such as

G = grades[7]**;**

The statement assigns the value stored in the 7th index of the array to the variable **g**. More generally if i is declared to be an integer variable, then the statement **g=grades[i];** Will take the value contained in the element number **i** of the grades array to assign it to g. so if **i** were equal to 7 when the above statement is executed, then the value of **grades[7]** would get assigned to g.

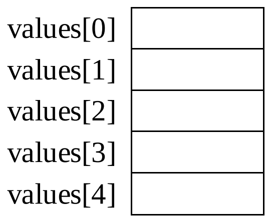
A value stored into an element in the array simply by specifying the array element on the left hand side of the equals sign. In the statement **grades [100]=95;** The value 95 is stored into the element number 100 of the grades array.

The ability to represent a collection of related data items by a single array enables us to develop concise and efficient programs. For example we can very easily sequence through the elements in the array by varying the value of the variable that is used as a subscript into the array. So the for loop

```
for(i=0;i < 100;++i){
        sum = sum + grades [i];
}
```

Will sequence through the first 100 elements of the array grades (elements 0 to 99) and will add the values of each grade into sum. When **for loop** is finished, the variable sum will then contain the total of first 100 values of the grades array.

The declaration int values[5]; would reserve enough space for an array called values that could hold up to 5 integers. Refer to the below given picture to conceptualize the reserved storage space.



Initialization of Arrays

We can initialize the elements in the array in the same way as the ordinary variables when they are declared. It can be initialized at either of the following stages.

- At compile time
- At run time

The general form of initialization of arrays is:

type array_name[size]={list of values};

The values in the list care separated by commas.

Compile Time Initialization

```
int number[3]={0,0,0};
```

Example#1:

```
/* A program to initialize character array at compile time */
#include<stdio.h>
#include<conio.h>
void main(){
        char name[] = {'R','a','m','\0'};
        char name2[] = "Khwopa";
        clrscr();
        printf("%s", name);
        printf("\n%s", name2);
        getch();
}
```

Run Time Initialization

An array can be explicitly initialized at run time.

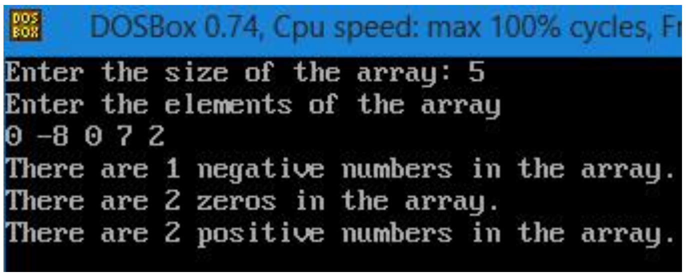
```
        int x[3];
        scanf("%d%d%d",&x[0],&x[1],&x[2]);
```

Accessing the Array Elements

i. Assigning values to array elements,
ii. Displaying the array elements

Example#2:

```
/* Program to count the no of negative, zero & positive numbers */
main(){
  int a[50],n,count_neg=0,count_zero=0,count_pos=0,I;  clrscr();
  printf("Enter the size of the array: ");  scanf("%d",&n);
  printf("Enter the elements of the array\n");
  for(I=0;I<n;I++){ scanf("%d",&a[I]);}
  for(I=0;I<n;I++){
    if(a[I]<0){ count_neg++;}
    else if(a[I]>0){ count_pos++;}
    else{ count_zero++;}
  }
  printf("There are %d negative numbers in the array.\n",count_neg);
  printf("There are %d zeros in the array.\n",count_zero);
  printf("There are %d positive numbers in the array.\n",count_pos);
  getch(); }
```



```
Enter the size of the array: 5
Enter the elements of the array
0 -8 0 7 2
There are 1 negative numbers in the array.
There are 2 zeros in the array.
There are 2 positive numbers in the array.
```

Two-dimensional Arrays

Often there is a need to store and manipulate two dimensional data structure such as matrices & tables. Here the array has two subscripts. One subscript denotes the row & the other the column. The declaration of two dimension arrays is as follows:

data_type array_name[row_size][column_size];

int m[10][20]

Here m is declared as a matrix having 10 rows(numbered from 0 to 9) and 20 columns(numbered 0 through 19). The first element of the matrix is m[0][0] and the last row last column is m[9][19]

Elements of Two-dimensional Arrays:

A 2-dimensional array **marks[4][3]** is shown below figure. The first element is given by marks [0][0] contains 88.2 & second element is marks [0][1] and contains 90.5 and so on.

marks[0][0] 88.2	marks[0][1] 90.5	marks[0][2] 89.5
marks[1][0]	marks[1][1]	marks[1][2]
marks[2][0]	marks[2][1]	marks[2][2]
marks[3][0]	marks[3][1]	marks[3][2]

Initialization of Two-dimensional Arrays:

Like the one dimension arrays, 2 dimension arrays may be initialized by following their declaration with a list of initial values enclosed in braces. For Example:

int table[2][3]={0,0,01,1,1};

Initializes the elements of first row to zero and second row to 1. The initialization is done row by row. The above statement can be equivalently written as

int table[2][3]={{0,0,0},{1,1,1}}

By surrounding the elements of each row by braces C allows arrays of three or more dimensions. The compiler determines the maximum number of dimension.

The general form of a ***multi-dimensional array*** declaration is:

date_type array_name[s1][s2][s3]…[sn]; where s is the size of the ith dimension.

Some examples are:

int survey[3][5][12];

float table[5][4][5][3];

Survey is a 3 dimensional array declared to contain 180 integer elements. Similarly table is a four dimensional array containing 300 elements of floating point type.

Example#3: Program to Print Multiplication Table

```
#define    ROWS     5
#define    COLUMNS  5
void main(){
    int   row, column, product[ROWS][COLUMNS] ;
    int   i, j ;
    printf("   MULTIPLICATION TABLE\n\n") ;
    printf("    ") ;
    for( j = 1 ; j <= COLUMNS ; j++ )
      printf("%4d" , j ) ;
    printf("\n") ;
    printf("-------------------------\n");
    for( i = 0 ; i < ROWS ; i++ ){
        row = i + 1 ;
        printf("%2d |", row) ;
        for( j = 1 ; j <= COLUMNS ; j++ ){
           column = j ;
           product[i][j] = row * column ;
           printf("%4d", product[i][j] ) ;
        }
        printf("\n") ;
    }
}
```

Output

```
     MULTIPLICATION TABLE

        1   2   3   4   5
    ----------------------
 1 |   1   2   3   4   5
 2 |   2   4   6   8  10
 3 |   3   6   9  12  15
 4 |   4   8  12  16  20
 5 |   5  10  15  20  25
```

Assignment:

Give an example program to add two matrices & store the results in the 3rd matrix.

Passing Array Element to a Function

Example#4:

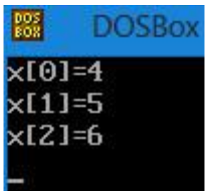
```
/* Passing Array Element to a function */
#include<stdio.h>
#include<conio.h>
void fxn_sum(int, int);
void main(){
             char numbers[] = {5,10};
             clrscr();
             fxn_sum(numbers[0],numbers[1]);
             getch();
}
void fxn_sum(int a, int b){
            printf("Sum = %d\n", (a+b));
}
```

Passing Array to a Function

An entire array can be passed to a function as an argument.

Example#5: Passing an Entire Array to a Function

```
#include<stdio.h>
#include<conio.h>
void fxn(int a[]);
void main(){
            int i, a[]={4,5,6}; clrscr();
            fxn(a);
            getch();
}
void fxn(int x[]){
            int i;
            for(i=0;i<3;i++){
              printf("x[%d]=%d\n",i,x[i]);
            }
}
```



```
x[0]=4
x[1]=5
x[2]=6
_
```

Passing 2-D Array to a Function

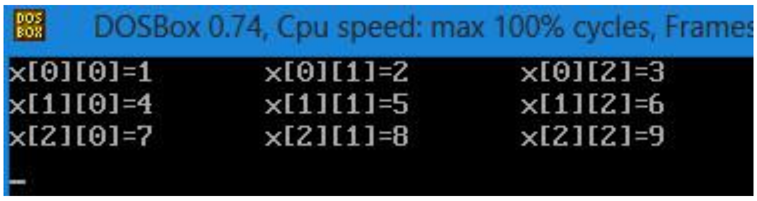
We can pass 2-D array same as 1-D array. Here is a Function Signature:

void fxn_display(int a[][]);

Example#6: Passing 2-D Array to a Function

```
#include<stdio.h>
#include<conio.h>
void fxn(int a[3][3]);
void main(){
    int i, a[3][3]={{1,2,3},{4,5,6},{7,8,9}};
    clrscr();
    fxn(a);
    getch();
}
void fxn(int x[3][3]){
    int i,j;
    for(i=0;i<3;i++){
        for(j=0;j<3;j++){
            printf("x[%d][%d]=%d\t",i,j,x[i][j]);
        }
        printf("\n");
    }
}
```

Output



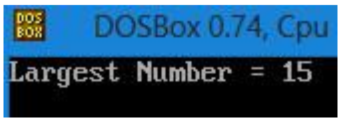
```
x[0][0]=1       x[0][1]=2       x[0][2]=3
x[1][0]=4       x[1][1]=5       x[1][2]=6
x[2][0]=7       x[2][1]=8       x[2][2]=9
_
```

Returning Array Element from a Function

Example#7:

```
/* Returning Array Element from a function */
#include<stdio.h>
#include<conio.h>
int find_largest(int a[]);
void main(){
        int numbers[] = {5,10,2,14,15};
        clrscr();
        printf("Largest Number = %d",find_largest(numbers));
        getch();
}
int find_largest(int x[]){
        int i;
        int largest = x[0];
        for (i=2;i<5;i++){
                if(largest<x[i]){
                        largest = x[i];
                }
        }
        return largest;
}
```

Output



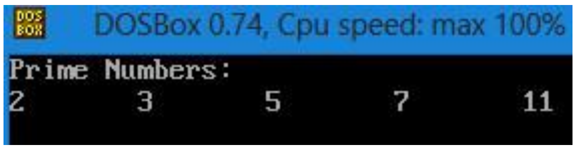
```
Largest Number = 15
```

Example#8: Returning Array from a Function

```
/* Returning Array from a function */
/* Program to generate prime numbers */
#include<stdio.h>
#include<conio.h>
int generate_prime(int *, int);
int is_prime(int);
void main(){
        int prime[100]; int i; int status=0;
        clrscr();
        status = generate_prime(&prime[0],5);
        printf("Prime Numbers:\n");
        for (i = 0; i<5 && status == 1; ++i){
                printf("%d\t", prime[i]);
        }
        getch();
}
int generate_prime(int *el, int n){
        int i, count=0;
        for (i=2; ; i++){
                if(is_prime(i)==1){
                        *el = i;
                        el++;count++;
                        if (count>=n){
                                break;
                        }
                }
        }
        return 1;
}
```

```
int is_prime(int n) {
   int i, countFactor = 0, result = 1;
   if (n < 2) {
          return 0;
   }

   for (i = 1; i < n && result == 1; i++) {
      if (n % i == 0) {
         countFactor++;
      }
      if (countFactor > 1) {
         result = 0;
      }
   }
   return result;
}
```

Output



```
Prime Numbers:
2         3         5         7         11
```

Strings

A string is a sequence of characters that is treated as a single data item.

printf("Welcome to Khwopa College of Engineering");

Declaration and Initializing String Variable

Syntax: char string_name[size];

char college[10] = "KHWOPA";

K	H	W	O	P	A	\0	\0	\0	\0

We can also declare the size much larger than the string size in the initializer. Computer creates array size of 10, places the value "KHWOPA" in it, terminates with the null character, and initializes all other elements to NULL. And the storage look like as shown above.

However, the following declaration is illegal.

char college[4] = "KHWOPA";

Reading Strings from Terminal

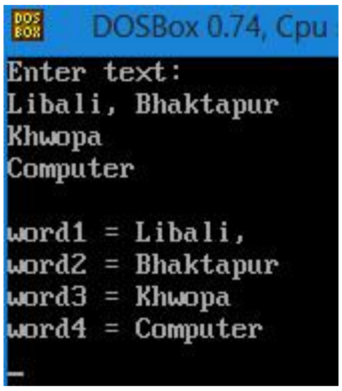
Using *scanf* Function

char address[20];

scanf("%s", adress);

Example#9:

```
#include<stdio.h>
#include<conio.h>
void main(){
        char word1[40], word2[40], word3[40], word4[40]; clrscr();
        printf("Enter text: \n");
        scanf("%s %s", word1, word2);
        scanf("%s", word3);
        scanf("%s", word4);
        printf("\nword1 = %s\nword2 = %s\n", word1, word2);
        printf("word3 = %s\nword4 = %s\n", word3, word4); getch();
}
```



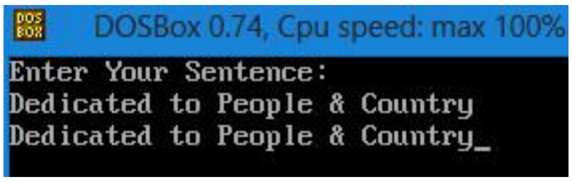
```
Enter text:
Libali, Bhaktapur
Khwopa
Computer

word1 = Libali,
word2 = Bhaktapur
word3 = Khwopa
word4 = Computer
_
```

Reading a Line of Text

Example#10:

```
#include<stdio.h>
#include<conio.h>
void main(){
        char line[100]; clrscr();
        printf("Enter Your Sentence:\n");
        scanf("%[^\n]",line);
        printf("%s", line); getch();
}
```

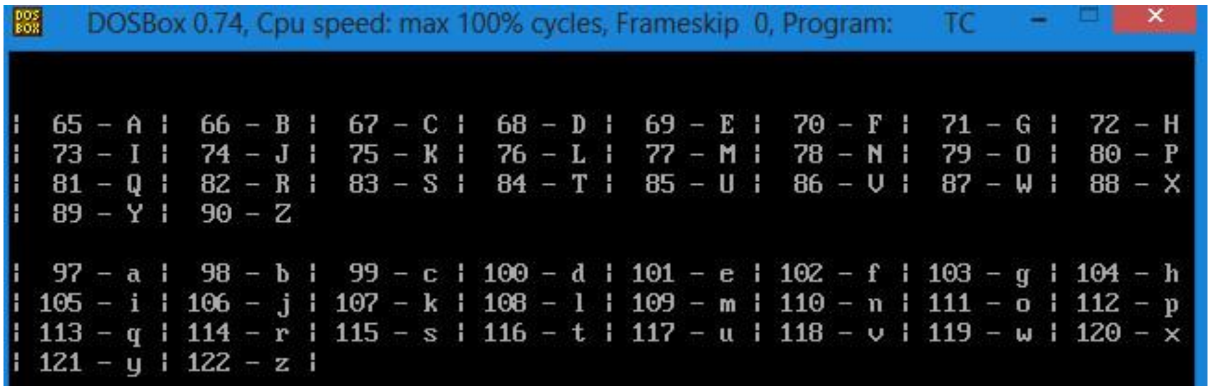
Output



```
Enter Your Sentence:
Dedicated to People & Country
Dedicated to People & Country_
```

Example#11:

```
#include<stdio.h>
#include<conio.h>
void main(){
        char c; clrscr(); printf("\n\n");
        for(c=65; c<=122; c++){
          if( c > 90  &&  c < 97 ){ continue;}
          if(c==97){ printf("\n\n");}
          printf("|%4d - %c ", c, c);
        }
        printf("|\n"); getch();
}
```



DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```


¦  65 - A ¦  66 - B ¦  67 - C ¦  68 - D ¦  69 - E ¦  70 - F ¦  71 - G ¦  72 - H
¦  73 - I ¦  74 - J ¦  75 - K ¦  76 - L ¦  77 - M ¦  78 - N ¦  79 - O ¦  80 - P
¦  81 - Q ¦  82 - R ¦  83 - S ¦  84 - T ¦  85 - U ¦  86 - V ¦  87 - W ¦  88 - X
¦  89 - Y ¦  90 - Z

¦  97 - a ¦  98 - b ¦  99 - c ¦ 100 - d ¦ 101 - e ¦ 102 - f ¦ 103 - g ¦ 104 - h
¦ 105 - i ¦ 106 - j ¦ 107 - k ¦ 108 - l ¦ 109 - m ¦ 110 - n ¦ 111 - o ¦ 112 - p
¦ 113 - q ¦ 114 - r ¦ 115 - s ¦ 116 - t ¦ 117 - u ¦ 118 - v ¦ 119 - w ¦ 120 - x
¦ 121 - y ¦ 122 - z ¦
```

String Manipulation

The C Library supports a large number of ***string-handling functions*** that can be used to carry out many of the ***string manipulations*** task.

Function	Action
strcat()	Concatenates two strings
strcmp()	Compares two strings
strcpy()	Copies one string over another
strlen()	Finds the length of a string

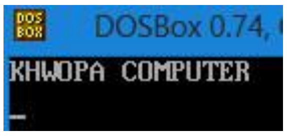
strcat() Function

The *strcat* function joins two strings together. It takes the following form:

```
strcat(string1, string2);
```

Example#12: String Concatenation

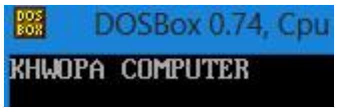
```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main(){
        char name1[] = {"KHWOPA "};
        char name2[] = {"COMPUTER"};
        clrscr();
        printf("%s\n", strcat(name1,name2)) ;
        getch();
}
```



```
KHWOPA COMPUTER
_
```

Example#13: String concatenation without using string.h

```
#include<stdio.h>
#include<conio.h>
void main(){
        int  i, j;
        char  name1[10] = {"KHWOPA"}  ;
        char  name2[10] = {"COMPUTER"} ;
        char   name[20] ;
        clrscr();
        /* Copy name1 into name */
        for(i=0; name1[i] != '\0'; i++ ){
          name[i] = name1[i] ;
        }
        /* End name1 with a space */
        name[i] = ' ' ;
        /* Copy name2 into name */
        for( j = 0 ; name2[j] != '\0' ; j++ ){
          name[i+j+1] = name2[j] ;
        }
        /* End name with a null character */
        name[i+j+2] = '\0' ;

        printf("%s\n", name) ;
        getch();
}
```



```
KHWOPA COMPUTER
```

strcmp() Function

The *strcmp* function compares two strings identified by the arguments and has a a value 0 if they are equal. If they are not, it has the numeric difference between the first non-matching characters in the strings. It takes the form:

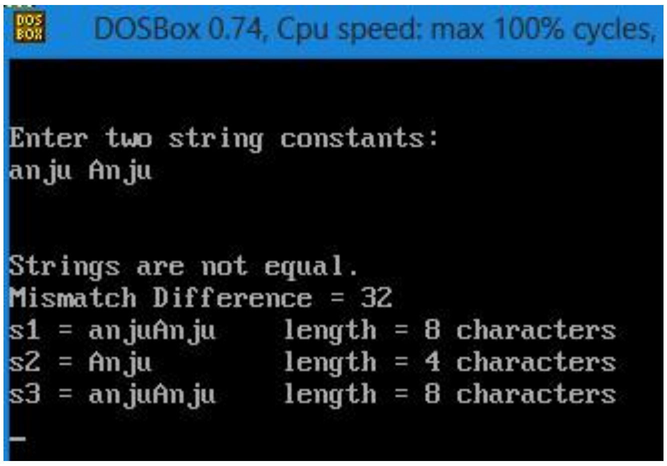
```
strcmp(string1, string2);
```

string1 & string2 may be string variables or string constants. Examples are:

```
strcmp(name1, name2);
strcmp(name1, “Ram”);
strcmp(“Sujan”, “Sujan”);
```

Example#14: Illustrations of String-Handling Functions

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main(){
        char s1[20], s2[20], s3[20];
        int   x, l1, l2, l3;
        clrscr();
        printf("\n\nEnter two string constants:\n");
        scanf("%s %s", s1, s2);
        /* comparing s1 and s2 */
        x = strcmp(s1, s2);
        if(x != 0){
          printf("\n\nStrings are not equal.\nMismatch Difference = %d",x);
          strcat(s1, s2);   /* joining s1 and s2, assign concatenated value in s1 */
        }
        else
          printf("\n\nStrings are equal \n");
        strcpy(s3, s1); /* copying s1 to s3 */
        /* Finding length of strings */
        l1 = strlen(s1); l2 = strlen(s2); l3 = strlen(s3);
        printf("\ns1 = %s\t length = %d characters\n", s1, l1);
        printf("s2 = %s\t length = %d characters\n", s2, l2);
        printf("s3 = %s\t length = %d characters\n", s3, l3);
        getch();
}
```



```


Enter two string constants:
anju Anju


Strings are not equal.
Mismatch Difference = 32
s1 = anjuAnju     length = 8 characters
s2 = Anju         length = 4 characters
s3 = anjuAnju     length = 8 characters
_
```

strcpy() Function

The *strcpy* function works almost like a string-assignment operator. It takes the form:

strcpy(string3, string1);

and assigns the contents of *string1* to *string3*. *string1* may be character variable or a string constant.

strcpy(string3, “Khwopa”);

strlen() Function

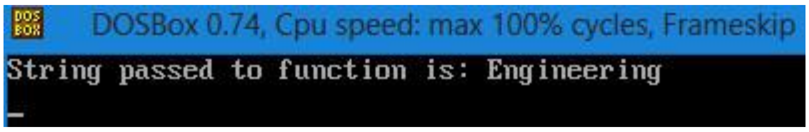
This function counts and returns the number of character in a string. It takes the form:

n = strlen(“Computer”); // n=8

Passing String to a Function

Example#15:

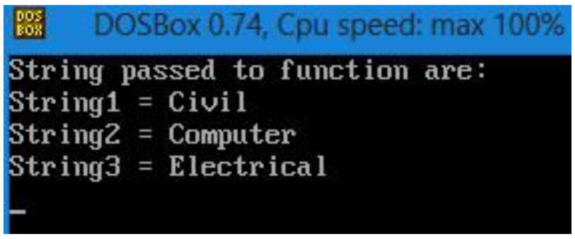
```
#include<stdio.h>
#include<conio.h>
void fun(char a[]){ printf("String passed to function is: %s\n", a);}
void main(){
        char text[] = {"Engineering"}; clrscr();
        fun(text); getch();
}
```



```
String passed to function is: Engineering
_
```

Example#16: Passing Multiple Strings to a Function

```
#include<stdio.h>
#include<conio.h>
void fun(char a[3][10]){
        int i; printf("String passed to function are:\n");
        for (i=0; i<3; i++){ printf("String%d = %s\n",i+1,a[i]);}
}
void main(){
        char text[3][10] = {"Civil","Computer","Electrical"}; clrscr();
        fun(text); getch();
}
```



```
String passed to function are:
String1 = Civil
String2 = Computer
String3 = Electrical
_
```