EST 102 CPC

MODULE-3

Arrays:

- Fundamental data types like char, int, float, double are very useful in programming.
- But they are constrained by the fact that a variable of these types can store only one value at any given time.
- So, only limited amounts of data can be handled using data types.
- For handling large volumes of data, a powerful data type that facilitate efficient storing, accessing and manipulation of data items are required.
- C supports a **derived data type** known as array that can be used in such applications.
- An array is a **fixed-size sequenced collection of elements of the same data type.**
- It is simply a grouping of like-type data.
- Array provides a convenient structure for representing data and it is one of the **data structures** in C.

- Eg: We can use an array named *salary* to represent a set of salaries of a group of employees in an organization.
  - By writing a number called *index* or *subscript* in brackets after the array name, we can refer to the individual salaries.

salary [10] represents the salary of the 10th employee.

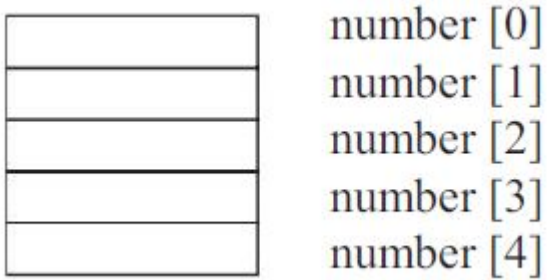
- The complete set of values is referred to as an ***array*** whereas the individual values are called ***elements***.
- We can use arrays to represent not only simple lists of values but also tables of data in two, three or more dimensions.
- In C, arrays are classified into;
  - One-dimensional arrays
  - Two-dimensional arrays
  - Multi-dimensional arrays

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.
- Eg: Suppose we want to represent a set of five numbers, say (1, 3, 5, 7, 9), by an array named ***number*****,** it can be done as follows;

  int number[5];

  and the computer reserves five storage locations as shown below:



- The values to the array elements can be assigned as follows;

```
number[0] = 1;
number[1] = 3;
number[2] = 5;
number[3] = 7;
number[4] = 9;
```

- This would cause the array ***number*** to store the values as shown below;

number[0]	1
number[1]	3
number[2]	5
number[3]	7
number[4]	9

Declaration of one-dimensional arrays:

- Arrays should be declared before using them in the C program.
- Declaration of arrays allows the compiler to allocate space for them in the memory.
- The general form of array declaration is as follows;

type variable-name[size];

- The ***type*** specifies the type of element 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.
- Eg: float height[50];

The above statement declares the height to be an array containing 50 real elements. So, subscripts from 0 to 49 are valid.

- Any reference to the arrays outside the declared limits would not necessarily cause an error. Rather, it might result in unpredictable program results.
- The size should be either a numeric or symbolic constant.
- C language treats character strings simply as array of characters. The *size* in a character string represents the maximum number of characters that the string can hold.
    - Eg: **char name[10];**

      The above statement declares the ***name*** as a character array (string) variable that can hold a maximum of 10 characters.

- Suppose we read the following string constant into the string variable ***name***.

  " WELL DONE "

- Each character of the string is treated as an element of the array name and is stored in the memory as follows;

'W'
'E'
'L'
'L'
' '
'D'
'O'
'N'
'E'
'\0'

- When the compiler sees a character string, it terminates it with an additional null character.
- Thus, the element name[10] holds the null '\0'.
- When declaring character arrays, we must allow one extra element space for the null terminator.

Initialization of one-dimensional arrays:

- After an array is declared, its elements must be initialized.
- Otherwise, they will contain “garbage”.
- An array can be initialized at either of the following stages:
  - At compile time
  - At run time

Compile Time Initialization:

- In compile time initialization, elements of the array are initialized whenever they are declared.
- The general form of initialization of arrays is:

type array-name[size] = { list of values };

- Eg:1

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

    The above statement will declare the variable *number* as an array of size 3 and will assign value zero to each element.

- Eg:2

    float total[5] = {0.0, 1.1, 2.22};

The above statement will declare the variable *total* as an array of size 5 and first three elements are initialized to 0.0, 1.1 and 2.22 and remaining two elements are set to 0.

char city[5] = {'B'};

The above statement will initialize the first element to 'B' and remaining four to NULL.

ie, **If the number of values in the list is less than the number of elements, then only that many elements will be initialized. The remaining elements will be set to zero automatically if the type is numeric and NULL if the type is char.**

- The *size* may be omitted.In such cases, the compiler allocates enough space for all initialized elements.

  Eg:**int counter[ ] = {1, 1, 1, 1};**

  The above statement will declare the *counter* array to contain 4 elements with initial values 1. As long as we initialize every element in the array, this method works fine.

- Character arrays may be initialized in a similar manner.

  char name[ ] = {'J', 'o', 'h', 'n', '\0'};

The above statement will declare the array named *name* to be an array of five characters, initialized with the string "John" ending with the null character.

Alternatively, we can use the following statement too.

char name[ ] = "John";

- If there are more initializers than the declared size, error is produced.

  Eg: **int number[3] = {1, 2, 3, 4, 5};**

Run Time Initialization:

- Array is explicitly initialized at the run time.
- This approach is usually applied for initializing large arrays.
- Consider the shown segment of a C program:

```
--------
--------
for (i = 0; i < 100; i = i+1)
{
   if    i < 50
         sum[i] = 0.0;
   else
         sum[i] = 1.0;
}
--------
--------
```

The first 50 elements of the array **sum** are initialized to 0.0 while the remaining 50 elements are initialized to 1.0 at run time.

C program to read n integers, store them in an array and find the sum and average:

```
#include <stdio.h>
int main()
{
   int A[100], n, i, sum = 0;
printf("Enter the number of elements you want to insert : ");
scanf("%d", &n);
for (i = 0; i< n; i++)
   {
printf("Enter element %d : ", i + 1);
scanf("%d", &A[i]);
   }

   for (i = 0; i< n; i++)
   {
     sum += A[i];
   }
printf("\nThe sum of the array is : %d", sum);
printf("\nThe average of the array is : %.2f", (float)sum / n);
   return 0;
}
```

C program to read n integers, store them in an array and search for an element in the array for Linear search:

```
#include<stdio.h>
main()
{
  int array[100], search, c, n;
  printf("Enter number of elements in array\n");
  scanf("%d", &n);

  printf("Enter %d integer(s)\n", n);

  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);

  printf("Enter a number to search\n");
  scanf("%d", &search);

  for (c = 0; c < n; c++)
  {
    if (array[c] == search)
    {
      printf("%d is present at location %d.\n", search, c+1);
      break;
    }
  }
  if (c == n)
    printf("%d isn't present in the array.\n", search);
}
```

Two-dimensional arrays:

- With one-dimensional arrays, a list of values can be stored.
- There can be situations where a table of values will have to be stored.
  - Eg: Consider the following data table, which shows the value of sales of three items by four sales girls:

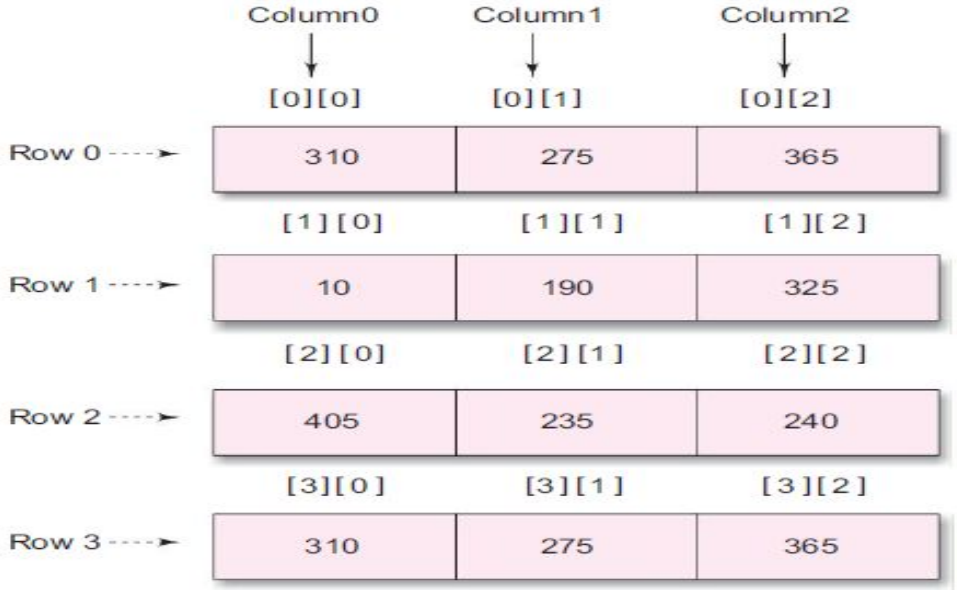
	Item1	*Item2*	*Item3*
Salesgirl #1	310	275	365
Salesgirl #2	210	190	325
Salesgirl #3	405	235	240
Salesgirl #4	260	300	380

- The table contains 12 values, three in each line. A table can be thought of as a matrix having 4 rows and 3 columns.
- **C allows us to define such table of items by using two-dimensional arrays.**

- Two-dimensional arrays can be declared as below:

  type array_name [row_size][column_size];

- Each dimension of the array is indexed from zero to its maximum size minus one; the first index selects the row and the second index selects the column within that row.
- Two dimensional arrays are stored in the memory as below;



Initializing two-dimensional arrays:

- Like one-dimensional arrays, two-dimensional arrays can be initialized by following their declaration with a list of values enclosed in braces.
- Eg:1

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

The above statement initializes the elements of the first row to zero and the second row to one.

- The initialization is done row by row.
- Eg1 can be rewritten as follows;

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

- A two-dimensional array can be initialized in the form of a matrix as below;

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

- When the array is completely initialized with all values, explicitly, we need not specify the size of the first dimension.
  - Eg: int table[ ][3] = {

    {0,0,0},

    {1,1,1}

    };

- If the values are missing in an initializer, they are automatically set to zero.
  - Eg: int table[2][3] = {

    {1,1},

    {2}

    };

  - The above statement the first two elements of the first row to one, first element of the second row to two, and all other elements to zero.

- To initialize all the elements to zero, the following statement can be used;
  - Eg: int m[3][5] = { {0}, {0}, {0}};

STRINGS:

- A string is **a sequence of characters that is treated as a single data item.**
- Any group of characters defined between double quotation marks is a string constant.
- If we want to include a double quote in the string to be printed, then we may use it with a backslash character as below,

  printf("\" Well Done ! "\");

  The above statement will output the string **" Well Done ! "**

- The statement

  printf("Well Done !");

  will output the string **Well Done !**

- Character strings are often used to build readable and meaningful programs.

- The common operations performed on character strings include:
    - Reading and writing strings.
    - Combining strings together.
    - Copying one string to another.
    - Comparing strings for equality.
    - Extracting a portion of a string.

Declaring and Initializing string variables:

- **C does not support strings as a datatype.**
- **It allows us to represent strings as character arrays.**
- In C, string variable is any valid C variable name and is always declared as an array of characters.
- The general form of declaration of a string variable is:

```
char string_name[ size ];
```

- The size determines the number of characters in the string_name.
  - Eg: char city[10];
- When the compiler assigns a character string to a character array, it automatically supplies a null character (‘\0’) at the end of the string.
- Therefore, the size should be equal to the maximum number of characters in the string plus one.
- Character arrays can be initialized when they are declared.
- C permits a character array to be initialized in either of the two following forms:

  char city [9] = “ NEW YORK “;

  char city [9] = {‘N’,’E’,’W’,’ ‘,’Y’,’O’,’R’,’K’,’\0’};

*The reason that city had to be 9 elements long is that the string NEW YORK contains 8 characters and one element space is provided for the null terminator.

- ie,When we initialize a character array by listing its elements, we must supply explicitly the null terminator.
- In C, we can also initialize the character array without specifying the number of elements.
- In such cases, size of the array will be automatically determined, based on the number of elements initialized.
  - Eg: char string [ ] = {'G','O','O','D','\0'};
  - The above statement defines the array named string as a five element array.
- We can also declare the size much larger than the string size in the initializer.
  - Eg: char string[10] = "GOOD";
  - The above statement creates a character array of size 10, places the value "GOOD" in it , terminates with the null character and initializes all other elements to NULL.
  - The storage will look like as below:
  -

G	O	O	D	\0	\0	\0	\0	\0	\0

- The following declaration is illegal;
  - Eg: char str2 [3] = “GOOD”;
  - This will result in a compile time error.
- In C, for strings, we can’t separate initialization from declaration.
  - Eg: char str2 [5];

    str2 = “GOOD”;

    This is invalid.

In-built string handling functions:

- The C library supports a large number of string handling functions that can be used to carry out many string manipulations.
- Following are some of the most common inbuilt string-handling functions used in C;
  - **strlen( )**
  - **strcat( )**
  - **strcpy( )**
  - **strcat( )**
  - **strcmp( )**
  - **puts( )**
  - **gets( )**

strcat( ) Function:

- **Joins two strings together.**
- It takes the following form;

strcat(string1, string2);

where string1 and string2 are character arrays.

- When ***strcat*** is executed, **string2 is appended to string1**.
- It does so by removing the null character at the end of string1 and placing string2 from there.
- The string at string2 remains unchanged.
- Care should be taken to ensure that the size of string1 (to which string2 is appended) is large enough to accommodate the final string.

- Eg1: Consider the execution of following statement in C;

  strcat(Part1, Part2);

  where, Part1 and Part2 are strings.

Execution of **strcat(Part1, Part2);** will result in:

- strcat function may also append a string constant to a string variable.
  - Eg2: **strcat(part1,”GOOD”);**
- C also permits nesting of strcat functions.
  - Eg3: **strcat(strcat(string1,string2), string3);**
  - Here, all the three strings are concatenated together and the resultant string is stored in string1.

Run | Debug | Stop | Share | Save | { } Beautify | Language C

main.c

```c
/*String handling Functions
C program to illustrate the use of strcat function in C*/
#include <stdio.h>
#include<string.h>
int main()
{
   char ch1[10]={'g', 'o', 'o', 'd', '\0'};
   char ch2[10]={'e','v','e','n','i','n','g','\0'};
   strcat(ch1,ch2);
   puts(ch1);
   return 0;
}
```

input

```
goodevening
```

strlen( ) Function:

- **Counts and returns the number of characters in a string.**
- It take the following form;

  n = strlen(string);

  Where **n** is an integer variable, which receives the value of the length of the **string** .

- The counting ends at the first null character.

Run | Debug | Stop | Share | Save | { } Beautify | Language C

main.c

```c
/*String handling functions
 strlen Function*/
#include<stdio.h>
#include <string.h>
int main()
{
  int n;
  char ch[20]={'H','E','L','L','O','\0'};
  n=strlen(ch);
  printf("Length of the string is: %d",n);
  return 0;
}

```

input

```
Length of the string is: 5
```

strcmp( ) Function:

- Compares two strings and takes value 0 if they are equal.
- If they are not equal, it has the numeric difference between the first non-matching characters in the strings.
- It takes the form:

strcmp(string1,string2);

where string1 and string2 can be string variables or string constants.

- Eg: strcmp(“their”, “there”);

The above statement will return a value -9 which is the numeric difference between ASCII “i” and ASCII “r”.

- If the value is negative, string1 is alphabetically above string2.

C program to compare two strings:

main.c

Ctrl+S

```c
/*String handling functions
 strcmp() function */
#include<stdio.h>
#include <string.h>
int main()
{
  char str1[20],str2[20];
  printf("Enter 1st string: ");
  gets(str1);
  printf("Enter 2nd string: ");
  gets(str2);
  if(strcmp(str1,str2)==0)
      printf("Strings are equal");
  else
      printf("Strings are not equal");
 return 0;
}
```

strcpy( ) function:

- **Copies one string over another.**
- It takes the form;

  strcpy(string1,string2);

- The contents of string2 is assigned to string1.
- string2 can be a character array variable or a string constant.
- Eg:
  - Consider the following statement;

    strcpy(city, “DELHI”);

    This will assign the string “DELHI” to the string variable city.

- Care should be taken to ensure that the size of string1 is large enough to receive the contents of string2.

Run | Debug | Stop | Share | Save | { } Beautify | Language C

main.c

```c
/*String handling functions
strcpy() function*/
#include<stdio.h>
#include <string.h>
int main()
{
 char ch1[20]="GOOD";
 char ch2[20];
 strcpy(ch2,ch1);
 printf("Value of second string is: %s",ch2);
 return 0;
}

```

input

```
Value of second string is: GOOD
```

gets:

- ***gets*** is a library function available in the <stdio.h> header file.
- Used to read a string of text containing whitespaces.
- It has the following form;

gets(str);

- **It reads characters into *str* from the keyboard** until a newline character is encountered and then appends a null character to the string.
- Eg:

```
char line[80];

gets (line);
```

puts:

- ***puts*** is a library function available in the <stdio.h> header file.
- Used to print string values.
- It has the following form;

    puts(string);

- *str* is a string variable containing a string value.
- This prints the value of the string variable *str* and then moves the cursor to the beginning of the next line on the screen.
- Eg:

```
char line[80];

gets(line);

puts(line);
```

Above program segment reads a line of text from the keyboard and displays it on the screen.