SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF CSE

23ITT101-PROGRAMMING IN C AND DATA STRUCTURES

I YEAR - II SEM

UNIT 3 – ARRAYS AND INTRODUCTION TO DATA STRUCTURES

TOPIC – One - 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**.
- The subscripted variable refers to the nth element of x.
- For example
- x[1], x[2], x[3],.....x[n]
- The subscript can begin with number 0. That is x[0] is allowed.

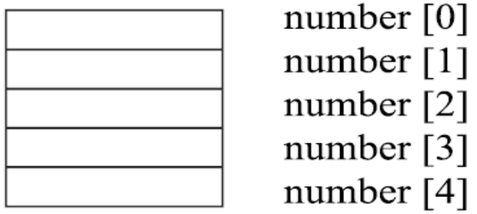
- For example, if we want to represent a set of five numbers, say (35, 40, 20, 57, 19), by an array variable number, then we may declare the variable number as follows

```
int number[5];
```

DECLARATION & STORAGE

- The five numbers to be stored in an one dimensional array is 35, 40, 20, 57, 19, where it can be declared as int number[5];

the computer reserves five storage locations as shown below:



The values to the array elements can be assigned as follows:

```
number[0] = 35;
number[1] = 40;
number[2] = 20;
number[3] = 57;
number[4] = 19;
```

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

number [0]	35
number [1]	40
number [2]	20
number [3]	57
number [4]	19

ARRAY - USAGE IN PROGRAM

- These elements (array) may be used in programs just like any other C variable.
- For example, the following are valid statements:

```
a = number[0] + 10;
number[4] = number[0] + number [2];
number[2] = x[5] + y[10];
value[6] = number[i] * 3;
```

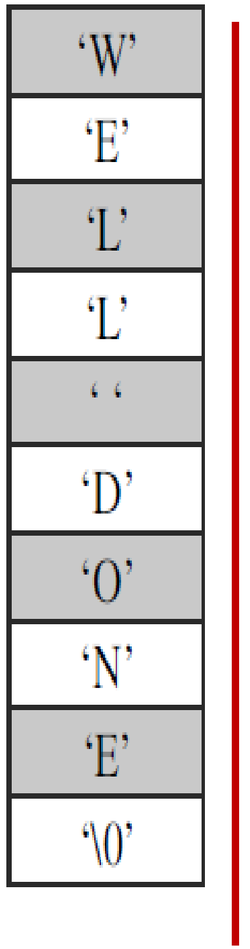
- C performs no bounds checking and, therefore, care should be exercised to ensure that the **array indices** are within the **declared limits**.
- For Example,
- int a[2] will support only 3 variables, if the value exceeds more than 3 means it will lead to error.

DECLARATION OF ONE-DIMENSIONAL ARRAYS

- Like any other variable, **arrays** must be **declared before** they are used so that the compiler can allocate space for them in memory.
- The general form of array declaration is type variable-name[size];
- The **type** specifies the type of element that will be contained in the array, such as **int, float, or char**
- 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.
- Similarly, int group[10];
  - declares the group as an array to contain a maximum of **10 integer constants**.
- Remember: Any reference to the arrays outside the declared limits would not necessarily cause an error.
- Rather, it might result in unpredictable program results.

DECLARATION OF ONE-DIMENSIONAL ARRAYS

- The C language treats **character strings** simply as arrays of characters.
- The size in a character string represents the maximum number of characters that the string can hold.
- For instance, char name[10];
  - 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 given in the figure.
- When the compiler sees a character string, it terminates it with an additional null character.
- Thus, the element name[10] holds the null character '\0'.
- NOTE:
- When declaring character arrays, we must allow one extra element space for the null terminator.



COMPILE TIME INITIALIZATION VS RUN TIME INITIALIZATION

Assigning or initializing the values for the variables while coding itself depicts compile time initialization.

```
#include <stdio.h>
int main()
{
    int a=20;
    printf("The value of a is : %d",a):
    return 0;
}
```

Assigning or initializing the values for the variables while getting the input from the user depicts run time initialization.

```
#include <stdio.h>
int main()
{
        int a;
        printf(“Enter The value of  a "):
        Scanf(“%d”, &a);
        printf("The value of a is : %d",a):
        return 0;
}
```

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:
  1. At compile time
  2. At run time

Compile Time Initialization

- We can initialize the elements of arrays in the same way as the ordinary variables when they are declared.
- The general form of initialization of arrays is: type array-name[size] = { list of values };
- The values in the list are separated by commas.
- For example, the statement int number[3] = { 0,0,0 };
- will declare the variable number as an array of size 3 and will assign zero to each element.

COMPILE TIME INITIALIZATION

- 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**.
- For instance, `float total[5] = {0.0,15.75,–10};`
- will initialize the **first three** elements to 0.0, 15.75, and –10.0 and the **remaining two** elements to **zero**.

- The **size** may be **omitted**.
- In such cases, the compiler allocates enough space for all initialized elements.
- For example, the statement `int counter[ ] = {1,1,1,1};`
- will declare the counter array to contain four elements with initial values 1.
- This approach works fine as long as we initialize every element in the array.

- **Character** arrays may be initialized in a similar manner.
- Thus, the statement `char name[ ] = {‘J’,‘o’, ‘h’, ‘n’, ‘\0’};`
- declares the name to be an array of five characters, initialized with the string “John” ending with the null character.
- Alternatively, we can assign the string literal directly as under: `char name [ ] = “John”;`

COMPILE TIME INITIALIZATION

- Compile time initialization may be partial.
- That is, the number of initializers may be less than the declared size.
- In such cases, the remaining elements are inilialized to zero, if the array type is numeric and
- NULL if the type is char.
- For example, int number [5] = {10, 20};
- will initialize the first two elements to 10 and 20 respectively, and the remaining elements to 0.
- Similarly, the declaration. char city [5] = {‘B’};
- will initialize the first element to ‘B’ and the remaining four to NULL.
- It is a good idea, however, to declare the size **explicitly**, as it allows the compiler to do some error checking.
- Remember, however, if we have more initializers than the declared size, the compiler will produce an error.
- That is, the statement int number [3] = {10, 20, 30, 40};
- will not work. It is illegal in C.

RUN TIME INITIALIZATION

- An array can be explicitly initialized at run time.
- This approach is usually applied for initializing large arrays
- For example, consider the following segment of a C program.

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

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

RUN TIME INITIALIZATION

```
// Program to take 5 values from the user and store them in an array
// Print the elements stored in the array
#include  <stdio.h>
#include <conio.h>
int main()
{
    int values[5];
    printf("Enter 5 integers: ");
    // taking input and storing it in an array
   for(int i = 0; i < 5; ++i)
           {
                   scanf("%d", &values[i]);
           }
    printf("Displaying integers: ");
   // printing elements of an array
     for(int i = 0; i < 5; ++i)
     {
            printf("%d\n", values[i]);
      }
    return 0;
}
```

OUTPUT:

```
Enter 5 integers: 1
-3
34
0
3
Displaying integers: 1
-3
34
0
3
```