C Programming

UNIT 3 Handouts / Class Notes

UNIT - 3 Part 1 (Arrays)

Dr. Suresh Mudunuri, SRKR Engineering College

UNIT 3 Syllabus

UNIT-III (10 Hrs)	Arrays: Concepts, Using Array in C, Array Application, Two Dimensional Arrays, Multidimensional Arrays, Programming Example – Calculate Averages Strings: String Concepts, C String, String Input / Output Functions, Arrays of Strings, String Manipulation Functions String/ Data Conversion, A Programming Example – Morse Code Enumerated, Structure, and Union: The Type Definition (Type def), Enumerated Types, Structure, Unions, and Programming Application.

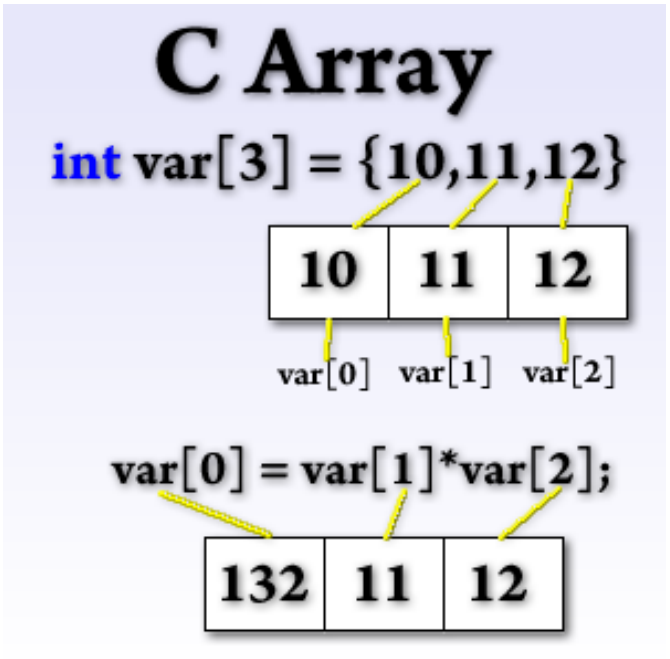
ARRAY

Array is a variable that can store multiple values of the **same data type**.

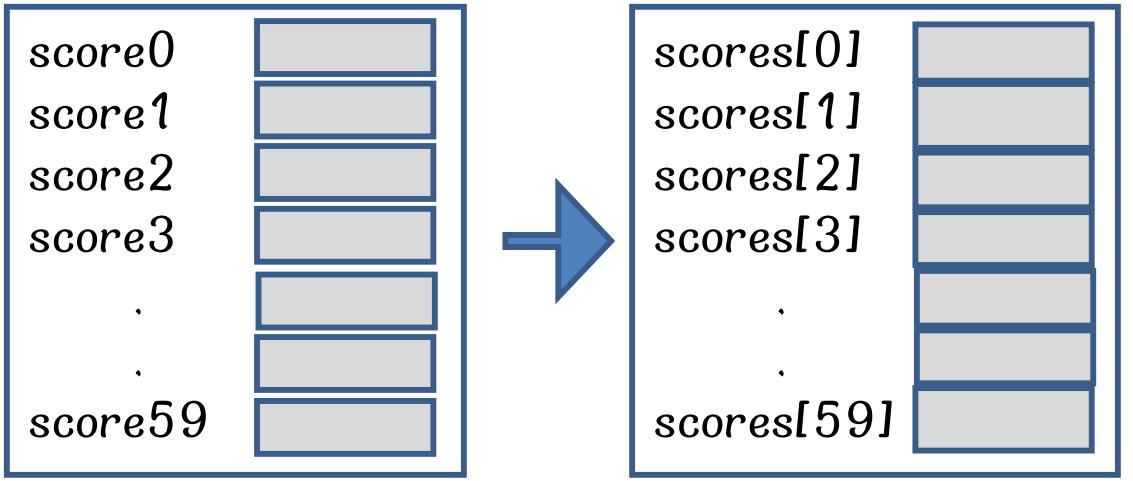
An *array* is a data structure that contains a group of elements.

Examples:

1D - int arr[10];
2D - int arr[10][10];
3D - float arr[3][10][5];
And so on...



Array & its Advantage

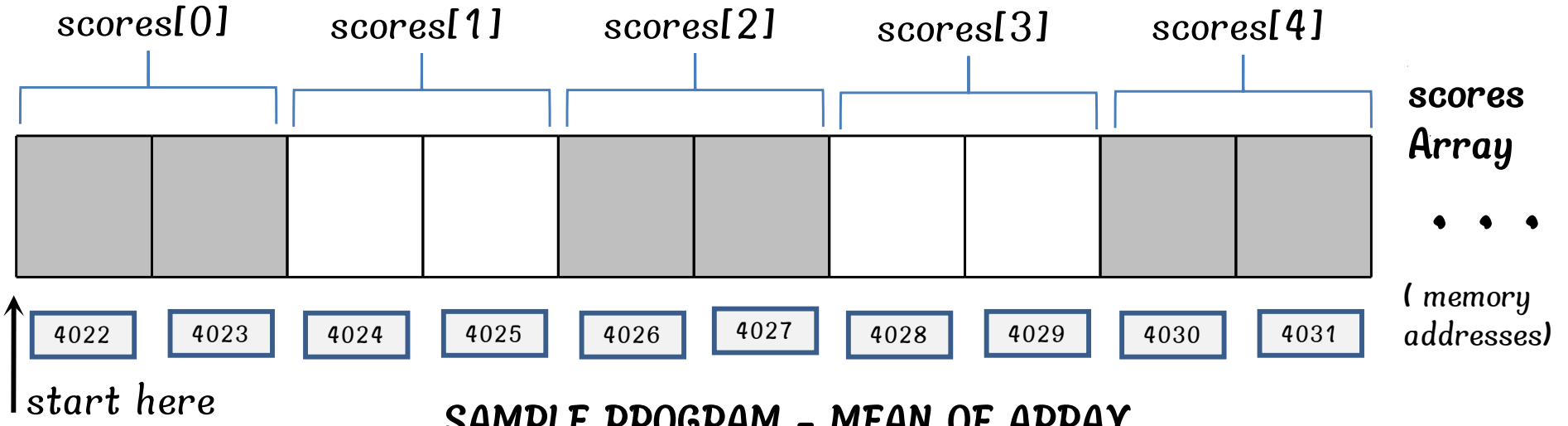


Sixty variables are replaced by one **Array**

```
#include<stdio.h>
int main() {
    int scores[60], i , j, temp;
    for(i = 0; i < 60 ;i++) {
        printf("Enter the score : ");
        scanf("%d", &scores[i]);
    }
    for(i=0;i<(60-1);i++)
      for( j=0; j <(60 -(i+1)); j++)
        if(scores[ j ] < scores[ j +1]) {
            temp = scores[ j ];
            scores[ j ] = scores[ j +1];
            scores[ j + 1] = temp;
        }
    for( i = 0; i  < 60; i ++) printf("%4d", scores[i]);
}
```

Sixty input statements are called by one loop statement

1770 comparing statements are included in one loop statement



SAMPLE PROGRAM - MEAN OF ARRAY

Mean can be calculated only after reading all scores. Each deviation is difference of individual score and mean. To calculate deviations of all scores, scores must be stored in an **ARRAY**.

```
#include<stdio.h>
#include<math.h>
#define SIZE 10
int main() {
   int scores[SIZE],sum=0,i;
   float deviation, mean, total=0;
   float variance , stddev;
   for(i=0; i < SIZE ;i++) {
     printf("Enter score : ");
     scanf("%d",   &scores[i] );
     sum = sum + scores[i];
   }
   mean = (float)sum / SIZE;
   printf("\nMean : %.2f",mean);
    printf("\nDeviations : ");
    for(i=0;i<SIZE ; i++) {
       deviation=   scores[i]    - mean;
       printf("%.2f\t", deviation);
       total=total + deviation*deviation;
    }
   variance = total / (SIZE-1);
   printf("\nVariance = %.2f\n", variance
);
   stddev = sqrt(variance);
printf("Standard Deviation : %f", stddev);
}
```

Declaration of Array

Initialization of Array

Finding Mean

Accessing an element

Processing on Array

Scalar variable for single data item & Vector variable for multiple data items

Scalar Variables :

➢ A variable represents a data item and it can be used to **store a single atomic value** at a time. These are also called scalar variables.

➢ Integer takes 2 bytes memory as a single unit to store its value. i.e.,the value of a scalar variable cannot be subdivided into a more simpler data items.

➢ The address of first byte is the address of a variable .

Vector Variables (arrays):

➢ In contrast, an **array** is multivariable (an aggregate data type), which is also referred to a **data structure**. It represents a collection of related data items of same type.

➢ An individual data item of an array is called as 'element'. Every element is accessed by **index or subscript** enclosed in square brackets followed after the array name.

➢ All its elements are **stored in consecutive memory locations**, referred under a common array name.

Ex : int marks[10] ; /* declaration of array */

➢ '0' is first number in computer environment. The first element of array marks is marks[0] and last element is marks[9]. (the address of first element is the address of the array)

➢ An array is a **derived data type**. It has additional operations for retrieve and update the individual values.

➢ The lowest address corresponds to the first element and the highest address to the last element.

➢ Arrays can have from one to several dimensions. The most common array is the ***string*, which is simply an array of characters terminated by a null**.

Declaration of One Dimensional Arrays

Syntax :

```
arrayType  arrayName [numOfElements ];
```

Example :

```
int scores [60];
float salaries [20];
```

Initialization of Array while Declaration :

```
int numbers [ ] = { 9, 4, 2, 7, 3 };
char  name[ ] ={'S','U','R','E','S','H','\0' };
char  greeting[ ] =  "Good Morning";
```

Declaration of Multi Dimensional Arrays

Syntax :

```
arrayType  arrayName [ Rows ][ Columns ];
arrayType  arrayName [ Planes][ Rows ][ Columns ];
```

Example :

```
/* Each student for seven subjects */
int marks[60][7];
/* Matrix with 3 planes and 5 rows and 4 columns */
float matrix[3][5][4];
```

Initialization of Array while Declaration :

```
int  matrix [ ][ ] = {  {  4, 2, 7, 3 } ,
                        {  6, 1, 9, 5 } ,
                        {  8, 5, 0, 1 }  };
```

Elements of
Array [3] by [4]

[0][0]	[0][1]	[0][2]	[0][3]
[1][0]	[1][1]	[1][2]	[1][3]
[2][0]	[2][1]	[2][2]	[2][3]

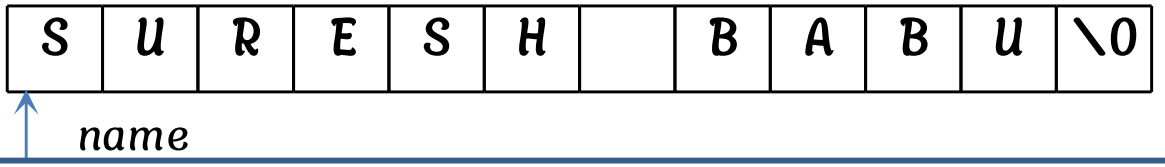
```
/* STORING A MATRIX IN 2D ARRAY */
int main()
{
    int matrix[5][5],i,j;

    for( i = 0; i  < 5 ; i++)
      {
        for(j=0;j<5;j++)
         {
            matrix[i][j]=0;
         }
      }
    return  0;
}
```

Strings - One Dimensional Character Arrays

A String is sequence of characters. In 'C' strings are implemented by an array of characters terminated with a null character '\0' (back slash followed by zero).

```
char name[] = "SURESH BABU";
```



'name' is an array of characters has size of 12 characters including a null character '\0'(ascii code is zero).

```
char name[25] ;
scanf("%s", name); /*reading a string until a white space is encountered ( & operator is not required )*/
printf("%s", name);  /*printing a string in input window */
gets(name) ; /* reading a string including white spaces until '\n' is encountered. */
puts(name); /* printing a string and moves cursor to new line */
```

String Manipulation Functions in <string.h>

strlen(s1)	- returns the length of string excluding the last 'null' character.
strcpy(s1,s2)	- copies characters in s2 into s1.
strcat(s1,s2)	- concatenates s2 to s1.
strcmp(s1,s2)	-compares s1 with s2 lexicographically and returns '0' if two strings are same , returns -1 if s1 is before s2 and returns +1 if s1 is after s2.
strcmpi(s1,s2)	-compares s1 with s2 like strcmp() but case of characters is ignored.
strchr(s1,ch)	-returns pointer to first occurrence of the character 'ch' in s1.
strstr(s1,s2)	-returns pointer to first occurrence s2 in s1.
strrev(s1)	-returns pointer to the reversed string.