2.1 Arrays

Objectives

- To define an array, initialize an array and refer to individual elements of an array.
- To define symbolic constants.
- To define and manipulate multiple-subscripted arrays.

Agenda

- Introducing Arrays
- Declaring Array Variables
- Initializing Arrays
- Accessing Array Elements
- Copying Arrays
- Multidimensional Arrays

Introduction

- An array is a collection of elements of the same type that are referenced by a common name.
- Compared to the basic data type (int, float & char) it is an aggregate or derived data type.
- All the elements of an array occupy a set of contiguous memory locations.
- Why need to use array type? ….. Consider the following issue:

"We have a list of 1000 students' marks of an integer type. If using the basic data type (int), we will declare something like the following…"

```
int  studMark0, studMark1, studMark2, ...,
                studMark999;
```

Introduction cont…

- Declaration part by using normal variable declaration

```
int main(void)
{
int studMark1, studMark2, studMark3, studMark4, …,
  …, studMark998, stuMark999, studMark1000;
…
…
return 0;
}
```

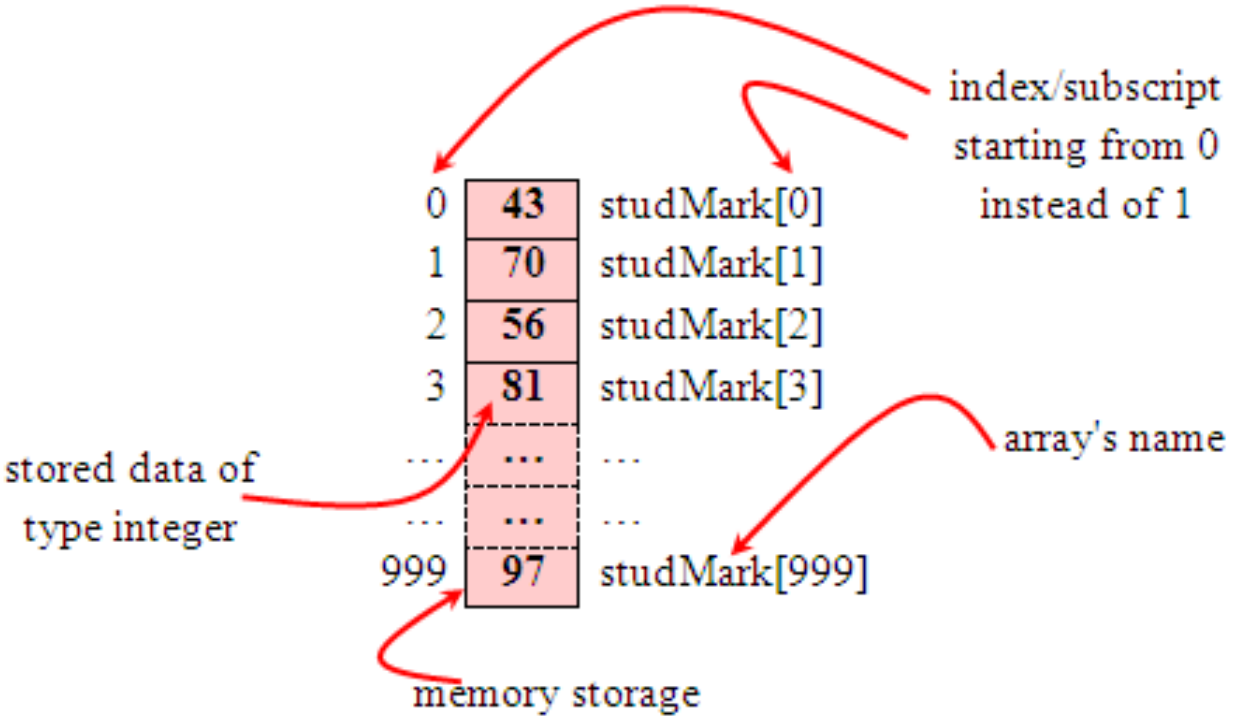
- By using an array, one can declare like this,

```
int  studMark[1000];
```

- This will reserve 1000 contiguous memory locations for storing the students' marks.

Introduction cont…

- Graphically, this can be depicted as



- So… array has simplified our declaration and of course, manipulation of the data.

One Dimensional Array: Declaration

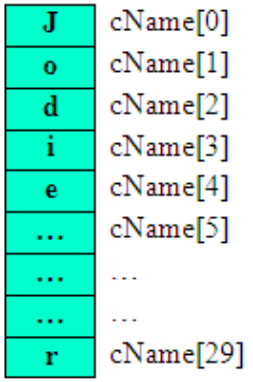
- Dimension refers to the array's size, which is how big the array is.
- A single or one dimensional array declaration has the following form,

```
array_element_data_type array_name[array_size];
```

- Here, *array_element_data_type* define the base type of the array, which is the type of each element in the array.
- *array_name* is any valid C identifier name that obeys the same rule for the identifier naming.
- *array_size* defines how many elements the array will hold.

- For example, to declare an array of 30 characters, that construct a people name, we could declare,

```
char   cName[30];
```



- In this statement, the array character can store up to 30 characters with the first character occupying location cName[0] and the last character occupying cName[29].
- Note that the index runs from 0 to 29. In C, an index always starts from 0 and ends with array's (size-1).
- So, take note the difference between the array size and subscript/index terms.

- Examples of the one-dimensional array declarations,

```
int     xNum[20], yNum[50];
float   fPrice[10], fYield;
char    chLetter[70];
```

- The first example declares two arrays named xNum and yNum of type int. Array xNum can store up to 20 integer numbers while yNum can store up to 50 numbers.
- The second line declares the array fPrice of type float. It can store up to 10 floating-point values.
- fYield is basic variable which shows array type can be declared together with basic type provided the type is similar.
- The third line declares the array chLetter of type char. It can store a string up to 69 characters.
- Why 69 instead of 70? Remember, a string has a null terminating character (\0) at the end, so we must reserve for it.

Starting from a given memory location, the successive array elements are allocated space in consecutive memory locations.

Array a

- x: starting address of the array in memory
- k: number of bytes allocated per array element

– **a[i] ➔ is allocated memory location at address** **x + i*k**

One Dimensional Array: Initialization

Method 1--Initialization at the time of declaration

- Giving initial values to an array.
- Initialization of an array may take the following form,

  type array_name[size] = {a_list_of_value};

- For example:

```
int   idNum[7] = {1, 2, 3, 4, 5, 6, 7};
float fFloatNum[5] = {5.6, 5.7, 5.8, 5.9, 6.1};
char  chVowel[6] = {'a', 'e', 'i', 'o', 'u', '\0'};
```

- The first line declares an integer array idNum and it immediately assigns the values 1, 2, 3, …, 7 to idNum[0], idNum[1], idNum[2],…, idNum[6] respectively.
- The second line assigns the values 5.6 to fFloatNum[0], 5.7 to fFloatNum[1], and so on.
- Similarly the third line assigns the characters 'a' to chVowel[0], 'e' to chVowel[1], and so on. Note again, for characters we must use the single apostrophe/quote (') to enclose them.
- Also, the last character in chVowel is NULL character ('\0').

- Initialization of an array of type char for holding strings may take the following form,

  char array_name[size] = "string_lateral_constant";

- For example, the array chVowel in the previous example could have been written more compactly as follows,

  char chVowel[6] = "aeiou";

- When the value assigned to a character array is a string (which must be enclosed in double quotes), the compiler automatically supplies the NULL character but we still have to reserve one extra place for the NULL.

- For unsized array (variable sized), we can declare as follow,

  char chName[ ] = "Mr. Dracula";

- C compiler automatically creates an array which is big enough to hold all the initializer.

Different cases: Initialization

- To set every element to same value

```
int n[ 5 ] = { 0 };
```

- If array size omitted, initializers determine size

```
int n[] = { 1, 2, 3, 4, 5 };
```

  - 5 initializers, therefore 5 element array

```
int n[ 5 ] = { 1, 2, 3, 4, 5 };
```

  - If not enough initializers, rightmost elements 0
  - If too many syntax error

One Dimensional Array: Initialization

Method 2 – Set the values using loop

```
int main()
    {       int n[ 10 ];  // n is an array of 10 integers
          // initialize elements of array n to 0
       for ( int i = 0; i < 10; i++ )
          n[ i ] = 0;   // set element at location i to 0
    }
```

Array size

- Can be specified with constant variable (**`const`**)

```
const int size = 20;
```

- Constants cannot be changed
- Constants must be initialized when declared
- Also called named constants or read-only variables
- The ***sizeof*** operator can determine the size of an array (in bytes).

```
int a[10];
```

sizeof(a) = 40 (assuming each integer requires 4 bytes)

One Dimensional Array: Accessing array elements

Individual elements of the array can be accessed by using the array name followed by the element subscript enclosed in square brackets as follows:

array_name[subscript]

Notice that the array elements start from 0, not 1, so the first element of the a array is a[0] and the last element is a[size-1] where size is the number of element in the a array.

The following program demonstrates how to access elements of an array:

```
#include <stdio.h>
 int main()
{
  const int SIZE = 5;
  int a[SIZE],i;
  for(i = 0; i < SIZE; i++)
  {              a[i] = i;
                  printf("a[%d] = %d\n",i,a[i]);
   }
 }
```

One Dimensional Array: Copying Arrays

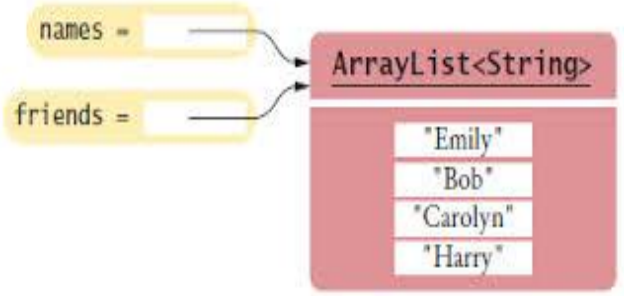
Can you copy array using a syntax like this?

list = myList;

This is not allowed in C.

You have to copy individual elements from one array to the other as follows:

```
for (int i = 0; i < ARRAY_SIZE; i++)
{
  list[i] = myList[i];
}
```



Rules to be followed when using arrays

- The data type can be any valid data type such as int, float, char, etc. [structure or union – Will be dealt in later chapter].
- All elements of an array must always be of the same data type
- The name of an array must follow naming rules of variables.
- The size of the array must be zero or a constant positive integer.
- The array index must evaluate to an integer between 0 and n-1 where n is the number of elements in the array.

Don't Do's

You cannot

- use = to assign one array variable to another

  a = b; /* a and b are arrays */

- use == to directly compare array variables

  if (a = = b) ……

- directly scanf or printf arrays

  printf ("……", a);

Illustrations

Summing Elements in an array

Use a variable named total to store the sum. Initially total is 0. Add each element in the array to total using a loop like this:

```
double total = 0;
for (int i = 0; i < ARRAY_SIZE; i++)
{
  total += myList[i];
}
```

Finding Maximum in an array

Use a variable named max to store the largest element. Initially max is myList[0]. To find the largest element in the array myList, compare each element in myList with max, update max if the element is greater than max.

```
double max = myList[0];
for (int i = 1; i < ARRAY_SIZE; i++)
{
  if (myList[i] > max) max = myList[i];
}
```

Finding index of the largest element in the array

```
double max = myList[0];
int indexOfMax = 0;
for (int i = 1; i < ARRAY_SIZE; i++)
{
 if (myList[i] > max)
 {
  max = myList[i];
  indexOfMax = i;
 }
}
```

Shifting Elements

```
double temp = myList[0]; // Retain the first element
// Shift elements left
for (int i = 1; i < myList.length; i++)
{
  myList[i - 1] = myList[i];
}
// Move the first element to fill in the last position
myList[myList.length - 1] = temp;
```

Try it Yourself - Predict the output


```
1. #include <stdio.h>
 int main()
{
   int arr[5];

   // Assume that base address of arr is 2000 and size of integer
      // is 32 bit
   arr++;
   printf("%u", arr);

   return 0;
}
```

(A) 2002

(B) 2004

(C) 2020

(D) lvalue required

Try it Yourself - Predict the output

2. What will be the output of the program ?

```
#include<stdio.h>
int main()
{ int a[5] = {5, 1, 15, 20, 25};
 int i, j, m; i = ++a[1];
j = a[1]++; m = a[i++];
printf("%d, %d, %d", i, j, m);
 return 0; }
```

A. 2, 1, 15

B. 1, 2, 5

C. 3, 2, 15

D. 2, 3, 20

Try it Yourself - Predict the output

3. What is the output of the following program?

```
int main()
{
   int i;
   int arr[5] = {0};
   for (i = 0; i <= 5; i++)
      printf("%d ", arr[i]);
   return 0;
}
```

A. Compiler Error: Array index out of bound.

B. The always prints 0 five times followed by garbage value

C. The program always crashes.

D. The program may print 0 five times followed by garbage value, or may crash if address (arr+5) is invalid.

Answers --- Predict the output

1. D) lvalue required

   Array name in C is implemented by a constant pointer. It is not possible to apply increment and decrement on constant types.

2. C) 3, 2, 15

   Step 1: *int a[5] = {5, 1, 15, 20, 25};* The variable arr is declared as an integer array with a size of 5 and it is initialized to

   a[0] = 5, a[1] = 1, a[2] = 15, a[3] = 20, a[4] = 25 .

   Step 2: *int i, j, m;* The variable i,j,m are declared as an integer type.

   Step 3: *i = ++a[1];* becomes *i = ++1;* Hence *i = 2* and *a[1] = 2*

   Step 4: *j = a[1]++;* becomes *j = 2++;* Hence *j = 2* and *a[1] = 3*.

   Step 5: *m = a[i++];* becomes *m = a[2];* Hence *m = 15* and *i* is incremented by 1(i++ means 2++ so i=3)

   Step 6: *printf("%d, %d, %d", i, j, m);* It prints the value of the variables *i, j, m*

   Hence the output of the program is 3, 2, 15

3. **D)** The program may print 0 five times followed by garbage value, or may crash if address (arr+5) is invalid.

Try it Yourself – Code debugging


```
#include <stdio.h>
#define MAXSIZE 10
void main()
{ int array[MAXSIZE];
  int i, num, negative_sum = 0;
  printf ("Enter the value of N \n");
  scanf("%d", &num);
  printf("Enter %d numbers \n", num);
 for (i = 0; i < num; i++)
 { scanf("%d", array[i]); }
/* Summation starts */
 for (i = 0; i < num; i++)
 { if (array[i] < 0) { negative_sum = negative_sum + array[i];
  }
 printf("\n Sum of all negative numbers = %d\n", negative_sum);
}
```

Answers – Code Debugging

```
#include <stdio.h>
#define MAXSIZE 10
void main()
{       int array[MAXSIZE];
        int i, num, negative_sum = 0;
        printf ("Enter the value of N \n");
        scanf("%d", &num);
        printf("Enter %d numbers \n", num);
        for (i = 0; i < num; i++)
        {
          scanf("%d", &array[i]);
        }    /* Summation starts */
         for (i = 0; i < num; i++)
          {
            if (array[i] < 0) { negative_sum = negative_sum + array[i];
          }
         printf("\n Sum of all negative numbers = %d\n", negative_sum);
}
```

Try it Yourself - Simple word problems

To Print the Alternate Elements in an Array Array - NH-WP3.c

Find 2 Elements in the Array such that Difference between them is Largest Array - NH-WP1.c

To Sort the Array in an Ascending Order Array - NH-WP2.c

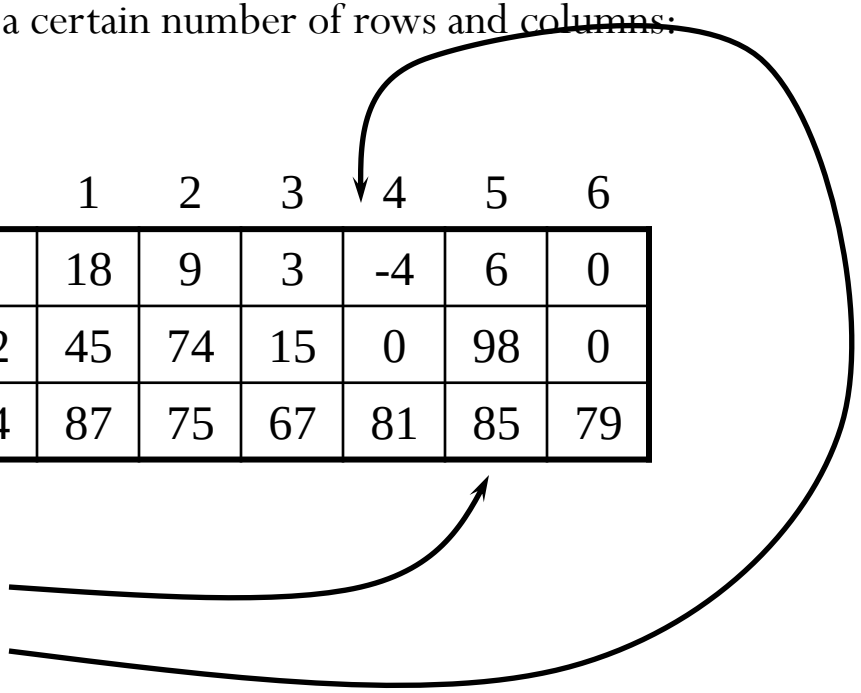
Multidimensional Arrays

- C also allows an array to have more than one dimension.

For example, a two-dimensional array consists of a certain number of rows and columns:

```
const int NUMROWS = 3;
const int NUMCOLS = 7;
int Array[NUMROWS][NUMCOLS];
```

	0	1	2	3	4	5	6
0	4	18	9	3	-4	6	0
1	12	45	74	15	0	98	0
2	84	87	75	67	81	85	79



`Array[2][5]` 3rd value in 6th column

`Array[0][4]` 1st value in 5th column

The declaration must specify the number of rows and the number of columns, and both must be constants.

Starting from a given memory location, the elements are stored row-wise

in consecutive memory locations.

x: starting address of the array in memory

c: number of columns

k: number of bytes allocated per array element

a[i][j] is allocated memory location at address

x + (i * c + j) * k

Multi Dimensional Array - Initialization

Method 1--Initialization at the time of declaration

```
int Array1[2][3]={{1, 2, 3},{4, 5, 6}};
int Array2[2][3] = { 1, 2, 3, 4, 5 };
int Array3[2][3] = { {1, 2} , {4 } };
```

Rows of Array1:

```
1 2 3
4 5 6
```

Rows of Array2:

```
1 2 3
4 5 0
```

Rows of Array3:

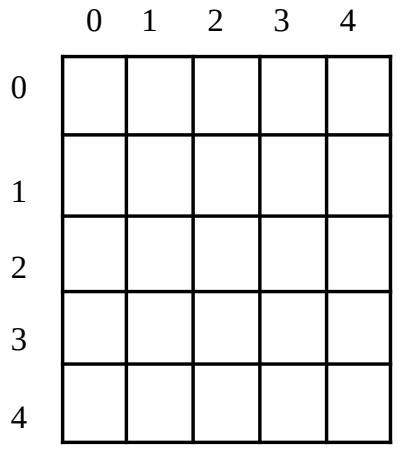
```
1 2 0
4 0 0
```

Multi Dimensional Array - Initialization

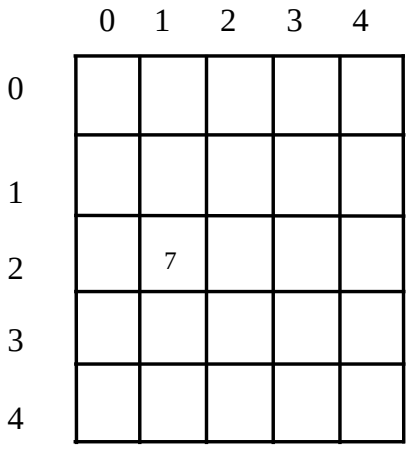
Method 2– Setting values using loop (nested loop)

```
int main()
{
const int NUMROW = 3;
const int NUMCOL = 7;
int Array1[NUMROW][NUMCOL];

for (int row = 0; row < NUMROW; row++)

{

   for (int col = 0; col < NUMCOL; col++)

{

    scanf("%d",&Array1[row][col];

   }

}
```

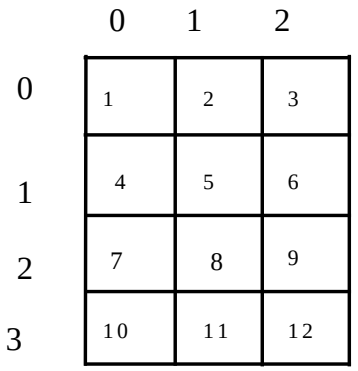
Multidimensional Array – Accessing Elements




```
matrix = new int[5][5];
```




```
matrix[2][1] = 7;
```




```
int[][] array = {
   {1, 2, 3},
   {4, 5, 6},
   {7, 8, 9},
   {10, 11, 12}
};
```

Multidimensional Array - Illustrations

To add two matrix entered by the user and print it.

```
#include<stdio.h>
void main() { int a[3][3],b[3][3],c[3][3];
int i,j;
printf("enter the elements in both the array:");
for(i=0 ; i<3 ; i++)
{
for(j=0 ; j<3 ; j++)
{
scanf("%d",&a[i][j]);
}
}
for(i=0 ; i<3 ; i++)
{
for(j=0 ; j<3 ; j++)
{
scanf("%d",&b[i][j]);
}
}
for(i=0 ; i<3 ; i++)
{
for(j=0 ; j<3 ; j++)
{
c[i][j]=a[i][j]+b[i][j];
printf("%d",c[i][j]);
}
printf("n");
}  }
```

Multidimensional Array - Illustrations

To input a matrix and print its transpose.

```
#include<stdio.h>
#include<conio.h>
void main()
{ int a[3][3],b[3][3];
 int i,j; clrscr();
printf("enter the elements in the array");
    for(i=0 ; i<3 ; i++)
{
for(j=0 ; j<3 ; j++)
 {
scanf("%d",&a[i][j]);
}
}
for(j=0 ; i<3 ; i++)
{
for(i=0 ; j<3 ; j++)
{
printf("%2d",&b[j][i]);
}
}
 getch();
 }
```

Try it Yourself - Predict the output

1. #include <stdio.h>

```
int main()
{
   int a[][] = {{1,2},{3,4}};
   int i, j;
   for (i = 0; i < 2; i++)
      for (j = 0; j < 2; j++)
         printf("%d ", a[i][j]);
   return 0;
}
```

A 1 2 3 4

B Compiler Error in line " int a[][] = {{1,2},{3,4}};"

C 4 garbage values

D 4 3 2 1

Try it Yourself - Predict the output

2. Consider the following declaration of a 'two-dimensional array in C:
   char a[100][100];

   Assuming that the main memory is byte-addressable and that the array is stored starting from memory address 0, the address of a[40][50] is.....?

A. 4040

B. 4050

C. 5040

D. 5050

Answers – Predict the Output

1. Answer: (B)

There is compilation error in the declaration

```
int a[][] = {{1,2},{3,4}};
```

Except the first dimension, every other dimension must be specified.

```
int arr[] = {5, 6, 7, 8} //valid
int arr[][5] = {}; //valid
int arr[][] = {}; //invalid
int arr[][10][5] = {}; //valid
int arr[][][5] = {}; //invalid
```

Answers – Predict the Output

2. Answer: (B)

Address of a[40][50] = Base address +

40*100*element_size +

50*element_size

= 0 + 4000*1 + 50*1 = 4050

Try it Yourself - Simple word problems

To Check if a given Matrix is an Identity Matrix MDArray-NH-WP!.c

To Calculate the Sum of the Elements of each Row & Column

MDArray-NH-WP2.c

Common Programming Errors

- It is important to note the difference between the "seventh element of the array" and "array element seven." Because array subscripts begin at 0, the "seventh element of the array" has a subscript of 6, while "array element seven" has a subscript of 7 and is actually the eighth element of the array. This is a source of "off-by-one" errors.
- Forgetting to initialize the elements of an array whose elements should be initialized.
- Providing more initializers in an array initializer list than there are elements in the array is a syntax error.
- Ending a `#define` preprocessor directive with a semicolon. Remember that preprocessor directives are not C statements.

- Assigning a value to a symbolic constant in an executable statement is a syntax error. A symbolic constant is not a variable. No space is reserved for it by the compiler as with variables that hold values at execution time.
- Not providing `scanf` with a character array large enough to store a string typed at the keyboard can result in destruction of data in a program and other runtime errors. This can also make a system susceptible to worm and virus attacks.
- Referencing a double-subscripted array element as `a[x,y]` instead of `a[x][y]`.

Summary

- The ability to use a single name to represent a collection of items and refer to an item by specifying the item number enables us to develop concise and efficient programs.
- C allows arrays of more than one dimensions.
- Exact limit is determined by the compiler