CHAPTER 5

STRUCTURED DATA TYPE - ARRAYS

5.1 Introduction

An array in C++ is a aderived data type that can hold several values of the same type.

Processing a collection of data values by reading the data items individually and then processing each item may be very cumbersome and awkward if data is large. For example, consider the following situations:

1. To determine the largest number in the given set of numbers:

   a) if the set comprises of two numbers then the comparisons would be :

```
if (a > b)
        max = a;
else
        max = b;
```

   b) if the set comprises of three numbers then the comparisons would be :

```
if (a > b) && (a > c)
        max = a;
else if (b > c)
        max = b;
else
        max = c;
```

c) if the given set comprises of 4 numbers then the comparisons would be :

```
if (a > b && a > c && a > d)
        max = a;
else if (b > c && b > d)
        max = b;
else if (c > d)
        max = c;
else
        max = d;
```

Have you noticed the increase in comparisons, as the numbers increase??

In fact, handling large data becomes unwieldy, if one has to adopt the above methods for processing data.

Now look at this:

```
int a [4] = { 10, 40, 30, 20}; max = 0 ; i = 0;
                for (; i < 4; i ++)
                if a [i] > max
                        max = a [i] ;
                cout << "\n The largest number is" << max;
```

The above program code determines the largest value in the given list of numbers, i.e., 10, 40, 30, 20, thus storing 40 in max. Have you noticed the construct of if statement? In order to handle large data with ease, elements belonging to the same data type are decaled as ARRAYS.

An array is a collection of variables of the same type that are referenced by a common name.

Arrays are of two types:

One dimensional: comprising of finite homogenous elements
Multi dimensional: comprising of elements, each of which is itself a one- dimensional array

5.2 Single Dimension Array

These are suitable ways for processing of lists of items for identical types. An array is declared as follows:

```
int num_array [5];
```

Syntax:

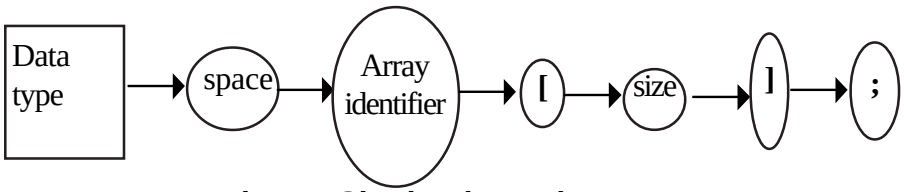


Fig. 5.1 Single Dimension Array

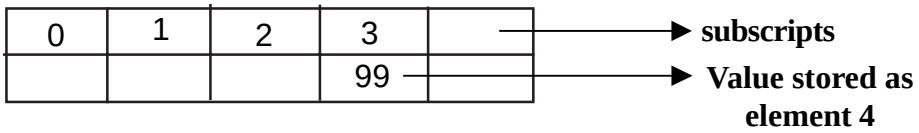
The size of the array should always be positive. The declaration int num_array [5]; is interpreted as **num_array is a one-dimensional array, that stores 5 integer values.** Each element of the array is accessed by the array name and the position of the element in the array. For example, num-array [3] = 99, stores the value 99 as the 4th element in the num_array.

num_array is the array identifier and **[3] is** subscript/position of the element

Memory allotted for num_array is 10 bytes, as it stores 5 integer element (memory required for one integer is 2 bytes hence 5 x 2 = 10 bytes).

Memory allocation is as follows:

num_array - identifier



The array subscripts always commences from zero, hence the subscripts for the variable num-array is between 0-4. The statement num-array [5] is invalid, because the valid subscripts are only between 0-4. The other valid examples of array declaration are:

i. int array [100];
ii. float exponents [10];
iii. char name [30];
iv. const i = 10;
double val [i];
v. int days [ ] = {1,2,3,4,5,6,7};

In example [iv], the size of the array val is indicated through a constant variable i. In example [v], the size of the array days [ ] is indirectly indicated as 7. Can you guess how? Yes, the size is determined through the initialization statement -{1,2,3,4,5,6,7}. 7 elements determine the size of the array. Now look at the Program – 5.1 that demonstrates basic operations on arrays.

Examples of array processing:

cin >> number [4]	-	Reads fifth element
cout << number [subscript]	-	displays the element as indicated by subscript
number [3] = number [2]	-	assigns the contents of the 3rd element of the array to its 4th element
number [3] ++	-	increments the value stored as 4th element by 1
number [++ a] = 10	-	assigns the value 10 to the element as indicated by ++a

```
// Program - 5.1
// arrays and basic manipulations on
// arrays
# include <iostream.h>
# include <conio.h>

int a[5],ctr = 0, sum;

void main()
{
    for(;ctr<5;ctr++)
   {
      cout << "\nEnter value ..";
      cin >> a[ctr];
   }
   // generating sum of all the elements
   // stored in array
   for(ctr = 0, sum = 0; ctr<5;ctr++)
     sum+= a[ctr];
   clrscr();
   // display values stored in array
   for(ctr = 0; ctr < 5; ctr++)
     cout <<'\t' << a[ctr];
   cout << "\nSum of the elements ..."
<< sum;
   getch();
}
```

```
// try out
// Program – 5.2

# include <iostream.h>

# include <conio.h>

char ch [ ] = {‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’};
void main ( )
{
      for (int i = 0; i < 5; i++)
           cout << ch[ i];

  for (j=4; j>=0; j—)
      cout << ch [j];

 getch();
}
```

Output diaplayed :
abcdeffedcba

```
// try out
// Program - 5.3
# include <iostream.h>
# include <conio.h>

void main ( )
{
int even [3] = {0, 2, 4}; int reverse [3];
for (int i=0, int j = 2; i<3; i++, j —)
      reverse [j] = even [i];
      clrscr ( );
for    (i=Ô;    i<3,    i++)
       cout ‘\t’ << even [i] << ‘\t’ << reverse [i] << ‘\n’;
       getch ( );
      }
```

Output of Program - 5.3

0	4
2	2
4	0

```
// try out
//Program - 5.4
# include <iostream.h>
# include <conio.h>
void main ( )
{
   int x[5] = {1,2,3,4,5}, y [5] = {5,4,3,2,1},
            result [5] = { 0,0,0,0,0 };
   int i= 0;
   while (i++ < 5)
            result [i] = x [i] - y [i];
   clrscr ( );
   cout << "\n The contents of the array are: \n";
   i= 0 ;
   do
   {
             cout << '\t' << x [i]
                      << '\t' << y [i]
                      << '\t' <<  result [i]<<'\n';
            i++;
   } while (i<5);
   getch ( );
}
```

Output of Program -5. 4

The contents of the array are:

1	-1	0
2	4	-2
3	3	0
4	2	2
5	1	4

```
//Try out
//Program - 5.5
# include <iostream.h>
# include <conio.h>
void main ( )
{
   int vector [ ] = {2, 4, 8, 10, 0};
   for(int i=4; i>2; i—)
       vector [i]= vector [i-1];
   clrscr( );
   cout << "\n Elements of array before insertion
\n";
   for (i= 0; i<5; i++)
      cout << vector[i];
   vector [2] = 6;
   cout << "\n Elements after insertion \n";
   for (i= 0; i<5; i++)
      cout << vector[i];
   getch ( );
}
```

Output of Program - 5.5

Elements of array before insertion
248810
Elements after insertion
246810

One can rearrange the data in a given array either in ascending or descending order. This process is called SORTING.

5.3 Strings

Strings are otherwise called as literals, which are treated as single dimensional array of characters. The declaration of strings is same as numeric array. For example,

i. char name [10];
ii. char vowels [ ] = {'a', 'e', 'i', 'o', 'u'};
iii. char rainbow [ ] = VIBGYOR;

A character array (used as string) should be terminated with a '\0' (NULL) character. These arrays can be initialized as in the above examples, viz., (ii) and (iii).

```
// Program - 5.6
// Reading values into an array of characters

# include <iostream.h>
# include <stdio.h>
# include <conio.h>
# include <string.h>
void main()
{
   clrscr();
   char name [30], address [30], pincode[7];
   cout << "\n Enter name ...";
   cin >> name;
   cout << "\n Enter address...";
   gets (address);
   cout << "\n Enter pincode ...";
   cin.getlin e (pincode, 7,'#');
   clrscr ( );
   cout << "\n Hello  " << name;
   cout << "\n Your address is ..." << address;
   cout << "\n Pincode  ....";
   cout.write (pincode, sizeof(pincode));
   getch ( );
}
```

In the above example Program - 5.6, the values for the variables name, address and pincode are read using

cin, gets ( ) and getline.

The instance cin, treats white space or carriage return (enter key) as terminator for string. For example,

cin >> name;

a) if the value for name is given as K V N Pradyot , then the value stored in name is only K, as white space is treated as string separator or terminator.
b) if the value for name is given as K.V.N.Pradyot , then the value stored in name is K.V.N.Pradyot.

To treat spaces as part of string literal, then one has to use gets ( ) defined in stdio.h or getline ( ) - a member function of standard input stream.

Syntax for gets ( ) is

gets (char array identifier) or
gets (char *)

Syntax for getline is

cin.getline (char*, no.of characters, delimiter);

There are two methods to display the contents of string.

1. cout << name - this is similar to any other variable.
2. cout.write (pincode, 7); or cout.write (pincode, size of (pincode));

write ( ) is a member function of standard output stream, i.e., ostream. All member functions of a class, should be accessed through an object /instance of class. The two parameters required for write () function are identifier string characters, and no. of characters to be displayed.

For example,

```
//Program - 5.7
# include <iostream.h>
# include <conio.h>
void main()
{
   clrscr ( );
   char name[] = "Tendulkar";
   int i=1;
   while (i<10)
   {
        cout.write (name, i);
        cout << '\n';
        i++;
   }
   getch ( );
}
```

Output

T
Te
Ten
Tend
Tendu
Tendul
Tendulk
Tendulka
Tendulkar

String manipulators defined in string.h are described in Table 5.1.

Sl.No.	Function	Syntax	Purpose & value returned
1	strlen ( )	strlen (char *)	Returns the number of characters stored in the array.For example, name =
2	strcpy ( )	strcpy (char *,	Copies source string to target string. For example,strcpy
3	strcmp ( )	strcmp (char	Compares the two given strings, and returns 0 if strings are equal, value >0, if string 1 is greater than string 2. Otherwise value less than 0. For example, strcmp (“Abc”, “Abc”) returns 0strcmp (“Abc”, “abc”) returns a value less than 0strcmp

Table 5.1 String Functions

Strings czn be manipulated element by element like a char array. For example,

```
// Program - 5.8
# include <iostream.h>
# include <conio.h>
void main ( )
{
    clrscr();
    char name[] = "Pascal", reverse[7];
    int i= 0;
    while (name[i] != '\0')
        i++;
    reverse[i] = '\0';
    –i;
    int j = 0;
    while (i>= 0)
        reverse [i–] = name [j++];
    cout << "\n The contents of the string are: "<< name;
    cout << "\n The contents of the reversed string ..."
         << reverse;
    getch ( );
}
```

What will be the output of the following program?

```
//Program - 5.9
# include <iostream.h>
# include <conio.h>
# include <string.h>
main()
{
   char word [ ] = "test";
   int i=0, k = 4, j = 0;
   clrscr( );
   while (i < 4)
   {
       j = 0;
       while (j<=i)
          cout << word [j++];
       cout << '\n';
       i++;
   }
  return 0;
}
```

5.4 Two-Dimensional Array

A two-dimensional array is an array in which each element is itself an array. For instance, an array marks [3] [4] is a table with 3 rows, and 4 columns.

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3

int makrs [3] [4] = {90, 70, 80, 0, 75, 95, 65, 0, 80, 90, 90, 0}; will create a table as;

	0	1	2	3
0	90	70	80	0
1	75	95	65	0
2	80	90	90	0

Note:

- √ The number of elements in a 2-dimensional array is determined by multiplying the number of rows with number of columns. In this example - The array marks has 12 elements.

- √ The subscripts always commence from zero. The subscript for rows is from 0 to 2, and for columns - 0 to 3.

- √ An element in a 2-D array is referred as Marks [Row] [Column]. For example, marks [0] [3] = marks [0] [0] + marks [0] [1] + marks [0] [2] will sum up the marks of the 1st row, viz., 90, 70, 80.

A 2-D array is declared as:

Type array-id [Rows] [Columns];

Example:

1. int a[3] [2]- declares 3 rows and 2 columns for the array a
2. const i=5;
   float num [i] [3] - declares a 2-D table num with 5 rows and 3 columns
3. short fine [‘A] [‘E’] - declares a 2-D table of 65 rows and 69 columns

Note:

The dimensions (rows/columns) of an array can be indicated

1. using integer constants
2. using const identifier of integer or ordinal
3. using char constants
4. using enum identifiers

5.4.1 Memory representation of 2-D arrays

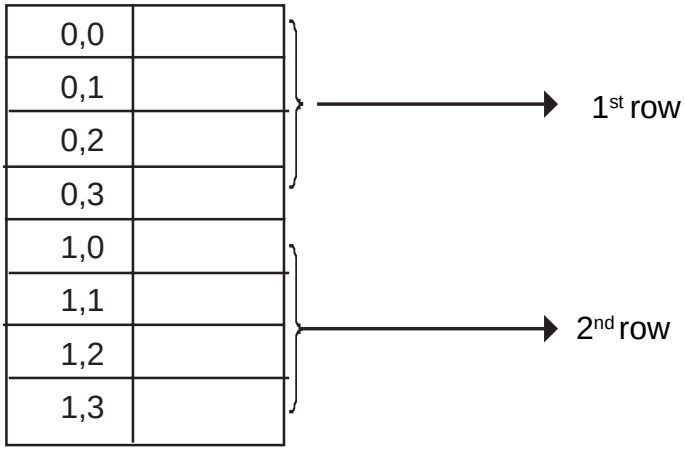
A 2-D array is stored in sequential memory blocks. The elements are stored either

1. row-wise manner (this method is called as row-major order)
2. column-wise manner (this method is called as column-major order)

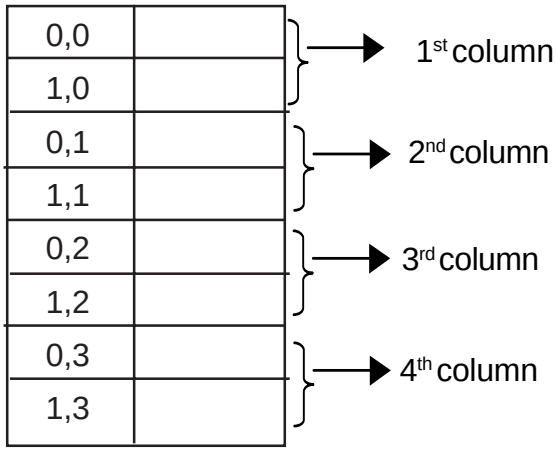
For example:

int sales [2] [4]; will be stored as follows:

In row-major order.



Column-major order



Have you noticed the position of sales [1] [0] in row-major order and column-major order?

The size of a 2-D array is calculated as follows:

Number of elements * memory req. for one element

For example - int sales [2] [4] the size will be calculated as follows:

Number of elements = Rows x columns - 2 x 4 = 8
∴ 8 x 2 (2 bytes is required for integer)
∴ size = 16 bytes

What will be the size of the array -

float num [4] [6];

Solution: 4 x 6 x 4 = 96 bytes

Consider the following array:

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

num

	0	1	2
0	8	**7**	6
1	4	**5**	8
2	**9**	7	6
3	1	**2**	3

Write the appropriate reference for the highlighted elements of the table num.

Solution:

num [0] [1] - (7)
num [1] [1] - (5)
num [2] [0] - (9)
num [3] [1] - (2)

Determine the number of elements in the following declaration:

a) int array [10] [12];
b) int x [ ] [2] = {0,1,1,2,2,3}

Solution:
a) 120 elements
b) 6 elements (rows = 3
columns = 2)

The size of first dimension (first index) is optional in array initialization.

```
#include <iostream.h>
# include <conio.h>
#include <iomanip.h>
void accept (int s [3] [4], int total)
{ int r = 0;  c = 0;
        for (; r < 3; r ++)
        {cout << "\n Month: " << r+1;
            for (; c < 4; c++)
                {cout << '\n' << c+1 <<
"Quarter..";
```

Arrays can be passed on as arguments to functions. The actual parameter is passed only by the identifier, ignoring dimensions.

Array parameters by default behave like a reference parameter, as the array identifier unlike other identifiers, represents the base address of the array. Hence, it results in sending an address to the formal parameter (like reference parameters).

```
// Program - 5.10
#include <iostream.h>
# include <conio.h>

void accept (int s[3][4], int &total)
{
   int r = 0, c = 0;
   for (; r < 3; r++)
   {
       cout << "\n Month: " << r+1;
       for (c = 0; c < 4; c++)
       {
            cout << '\n' << c+1 << " Quarter..";
            cin >> s[r][c];
            total += s[r][c];
       }
   }
}

void display (int d[3][4], int total)
{
   int r, c;
   clrscr ( );
   cout << "\nSales figures for 3 months & their
respective quarters..";
   for (r = 0; r < 3; r++)
   {
        cout << "\n Month ..." << r+1;
        for (c = 0; c < 4; c ++)
          cout << '\t' << d[r][c];
   }
   cout << "\n Total sales .." << total;
}
void main ( )
{
    clrscr();
    int sales[3][4], t = 0;
    accept(sales,t);
    display(sales,t);
    getch();
}
```

Now look at the following program.

```
// Program - 5.11
# include <iostream.h>
# include <conio.h>
void accept (int a)
{
   cout << "\n Enter a number ..";
   cin >> a;
}

void display (int a)
{
   cout << '\n' << a;
}

void main ( )
{
   int num [2][2] ={{0,0},{0,0}}, r = 0, c = 0;
   clrscr ( );
   for (; r < 2; r++)
   for (; c < 2; c++)
      accept (num[r][c]);
   clrscr();
   for (r = 0; r < 2; r++ )
    for (c = 0; c < 2; c++)
            display (num[r][c]);
   getch();
}
```

Output: Assume data entered in accept () function is 1,2,3,4

0
0
0
0

Why do you think the array num is not updated with the values 1,2,3,4?

In this example, the parameter passed to void accept ( ) is element by element. Hence, it is treated as value parameter and not reference parameter.

Note: Only the array identifier represents the base address of an array.

Now, rewrite the above program with the change - void accept (int (a). On execution, if the same test data 1,2,3,4 is given, then the output displayed will be

1
2
3
4

5.4.2 Matrix

A matrix is a set of mn numbers arranged in the form of a rectangular array of m rows and n columns. Matrices can be represented through 2-D arrays.

Program 5.12 demonstrates to read values for 2 matrices and check their equality.

5.5 Array of Strings

An array of strings is a two-dimensional character array. The size of first index (rows) determines the number of strings and the size of second index (column) determines maximum length of each string. For example,

```
char day-names [7] [10]     =     {“Sunday”,
                                  “Monday”,
                                  “Tuesday”,
                                  “Wednesday”,
                                  “Thursday”,
                                  “Friday”,
                                  “Saturday”};
```

will appear in the memory as shown in Table 5.1.

```
//Program - 5.12
# include <iostream.h>
# include <conio.h>

void accept (int mat[3][3])
{
   clrscr();
   int r = 0, c = 0;
   for (; r < 3; r++)
   {
       cout << "\n Enter elements for row.." << r;
       for (c=0; c < 3; c++)
            cin >> mat[r][c];
   }
}

void main ( )
{
    int m1[3][3], m2[3][3];
    accept (m1);
    accept (m2);
    int i=0, j = 0, flag = 1;
    for (; i < 3; i++)
    {
        for (; j < 3; j ++)
          if (m1[i][j] != m2[i][j])
          {
             flag = 0;
             break;
          }

          if (flag == 0)
             break;
   }
   if  (flag)
      cout << "\n The matrices are equal ...";
   else
      cout << "\n The matrices are not equal..";
  getch ( );
}
```

	0	1	2	3	4	5	6	7	8	9	
0	S	u	n	d	a	y	\0				day-names [0]
1	M	o	n	d	a	y	\0				day-names [1]
2	T	u	e	s	d	a	y	\0			day-names [2]
3	W	e	d	n	e	s	d	a	y	\0	day-names [3]
4	T	h	u	r	s	d	a	y	\0		day-names [4]
5	F	r	i	d	a	y	\0				day-names [5]
6	S	a	t	u	r	d	a	y	\0		day-names [6]

Table 5.1 Array Elements in Memory

An individual string is accessed as

day-names [0], i.e., by specifying the 1st index only. A specific character or an element is accessed as day-names [0] [5], i.e., by specifying both 1st and 2nd indices.

Attaching teh null character (\0) to each string literal is optional. Even if we omit it, the C++ compiler will automatically attach it.

Exercises

1. Why do the following snippets show errors?

a) int a [5.5]

Dimension of an array should be only an integer

b) float f [3] = {1.0, 2.0};

This will not show any error. But the number of elements is one less than the size of the array.

c) float num [A];

Dimension of an array should be explicitly mentioned. Here, the identifier A does not have a value. The statement may be rewritten as

```
float num ['A']
        Or
const A = 10;
float num [A];
```

d) `char a [3] [ ] = {"one", "two", "three"};`

The option for omitting the size of an array is given only for 1st index and not the second index. The statement may be rewritten as

`char a [ ] [6] = {"one", "two", "three"};`

e) `char ch [1] = 's';`

Character Array should be initialized using double quotes. The correct statement is

```
char ch [1] = "s"
        Or
char ch [1] = {"s"}
```

f)
```
char test [4];
test = {"abc"};
```

An array cannot be assigned in this manner. The correct statements are:

```
char test [4] = "abc" - initializing at the time of declaration
        or
char test [4];
strcpy (test, "abc");
```

g) int num [ ] = {1,2,3}, num2 [3];
num2 = num;

Group assignment of array is not allowed. One can assign only component by component.

h) int num [3];
cout << num;
cin >> num;

Such I/O operations are not allowed on arrays. Manipulation of arrays is possible only by specific direction to its elements or components, i.e.

cout << num [1] / cin >> num [1]

i) int roster = {1,2,3,4};

The variable roster cannot take more than one value. Hence, the statements should be as:

int roster = 10; or int roster [ ] = {1,2,3,4};

2. What would be the contents of the array after initialization?

a) int rate [ ] = {30,40,50};
b) char ch [6] = {" bbbb\0 " }
ch [0] = 'C';
ch [4] = 'T';
ch [3] = 'A';

Note b indicates white/blank space.

c) char product-list [ ] [5] = {"nuts", "Bolts", "Screw"};

Solution:

a - rate [0] = 30, rate [1] = 40. rate [2] = 50
b - ch [0] = 'C', ch [1] = ' '; ch [2] = ' ', ch [3] = 'A',
ch [4] = 'T', ch [5] = '\ 0",
c - product-list [0] = "Nuts \ 0", product-list [1] = "Bolts \ 0",
product-list [2] = "Screw\0"

3. What would be the output of the following programs?

a)
```
# include <iostream.h>
void main ( )
{char ch [ ] = {"END \ 0"S};
   cout << ch;
}
```
Solution: END

b)
```
# include <iostream.h>
void main ( )
{int a [ ] = {1,2,3,4,5};
for (int i = 0, i < 4, i++)
a[i+1] = a[i];
for (i= 0; i<5; i++)
cout << '\n' << a[i];
```

Solution:

	0	1	2	3	4
a	1	2	3	4	5

i = 0 - a [1] = a [0]

will be as follows:

i = 1 - a [2] = a [1]
i = 2 - a [3] = a [2]
i = 3 - a [4] = a [3]

	0	1	2	3	4	
a	1	2	3	4	5	

Hence, the array contents.

The output will be displayed as:

1
1
1
1
1

c)
```
# include <iostream.h>
# include <conio.h>
void main ( )
{ char name [ ] = {“Jerry \0”}; int k = 5;
        for (int i = 0 ;  i < 3; i ++, k —)
            name [k] = name [i];
         cout << name;
         getch ( );
}
```

Solution:

When i = 0 k = 5 name [5] = name [0]
i = 1 k = 4 name [4] = name [1]
i = 2 k = 3 name [3] = name [2]

∴ the output will be displayed as
JerreJ

Program Writing

1. Write a program to declare and initialize an array called as int-array, that stores number 10,20,30,40 and 50. Display the sum of all the elements of int-array.

2. Write a program to declare an array of integers that can hold 10 values. Read the elements of the array from the user, and also display the contents in the reverse order.

3. Write a program to read a sentence into an identifier called as word from the user. Using while loop and switch statements, display the count of vowels present in the given sentence. For example:

   word [ ] = “The vowel count AEIOU aeiou”;
   Vowel count is 14

4. Write a program to create a MATRIX [3] [3]. Display the diagonal elements along with the sum of diagonal elements.

5. Write a program to read values for two matrices, viz., matrix A [4] [4], matrix B [4] [4]. Write program code to create sum-matrix [4] [4] that stores the sum of elements of matrix A and matrix B.

Example

matrix A	**matrix B**	**sum-matrix**
1 2 3 4	9 8 7 6	10 10 10 10
5 6 7 8	5 4 3 2	10 10 - -
9 1 2 3	1 9 8 7	- - - -
4 5 6 7	6 5 4 3	- - - -