12 Arrays

CHAPTER

CHAPTER OUTLINE

12.1 INTRODUCTION

An array is a very popular, linear, homogenous, and useful data structure that is used to store similar types of data elements in contiguous memory locations under one variable name. An array can be declared to be of any standard or custom data type. The array of characters (strings) type works somewhat differently from the array of integers, floating numbers, and so on.

12.2 ONE-DIMENSIONAL ARRAY DECLARATION AND INITIALIZATION

The declaration of a one-dimensional array is as follows:

Declaration
`int a[5];`

It tells the compiler that `'a'` is an integer type of array, and it should store five integers. In this example, 5 is the subscript enclosed within square brackets. The compiler reserves two bytes of memory for each integer array element; that is, 10 bytes are reserved for storing five integers in the memory.

In the same way, arrays of different data types are declared as follows:

One-dimensional Array Declaration

```
char ch[10];
float real[10];
long num[5];
```

The array initialization is done as follows:

Array Initialization

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

Here, five elements are stored in an array *'a'*. The array elements are stored sequentially in separate locations. Then, the question arises of how to call each element individually from this bunch of integer elements. The reading of array elements begins from `zero`.

Array elements are accessed with the name of the array, and the number within the square brackets specifies the element number. In other words, array elements are called with array names followed by element numbers. Table 12.1 explains the accessing elements.

Table 12.1 Calling Array Elements

```
a[0] refers to 1st element i.e. 1
a[1] refers to 2nd element i.e. 2
a[2] refers to 3rd element i.e. 3
a[3] refers to 4th element i.e. 4
a[4] refers to 5th element i.e. 5
```

12.3 CHARACTERISTICS OF ARRAYS

(1) The Declaration `int a[5]` is nothing but a creation of five variables of integer types. Instead of declaring five variables for five values, the programmer can define them in an array.

(2) All the elements of an array share the same name, and they are distinguished from one another with the help of an element number.

(3) The element number in an array plays a major role for calling each element.

(4) Any particular element of an array can be modified separately without disturbing the other elements.

For example, `int a[5] = {1,2,3,4,8};`

If the programmer needs to replace 8 with 10, he/she is not required to change all the other elements except 8. To carry out this task, the statement `a[4] = 10` can be used. Here, the other four elements are left unchanged.

(5) Any element of an array `a[]` can be assigned/equated to another ordinary variable or an array variable of its type.

```
For example
b = a[2];
a[2] = a[3];
```

(a) In the statement `b = a[2]` or vice versa, the value of `a[2]` is assigned to `'b'`, where `'b'` is an integer.

(b) In the statement `a[2] = a[3]` or vice versa, the value of `a[3]` is assigned to `a[2]`, where both the elements are of the same data type.

(6) The array elements are stored in contiguous memory locations. The amount of storage required for holding the elements of the array depends on its type and size. The total size in bytes for a single-dimensional array is computed as shown below.

```
Total bytes = size of (data type) × size of array
```

The rules for array declaration are similar in both C and C++. Simple programs on arrays are demonstrated with C and C++.

12.1 Program executed in C

```
#include<stdio.h>
#include<conio.h>

void main()
{
   char text[3] = "xyz";
   char txt[3] = "abc";
   clrscr();
   printf ("\n %s",text);
}
```

OUTPUT

xyz_abcW__

12.2 Program executed in C++

```
#include<conio.h>
#include<iostream.h>

void main()
{
   char text[3] = "xyz";
   char txt[3] = "abc";
   clrscr();
   cout<<text;
}
```

OUTPUT

XyzÅ

Explanation: Both the above programs are similar except for the syntax. The first and second programs are executed in C (`Ver 2.0`) and C++ (`ver 3.0`), respectively.. The character arrays are declared to be exactly equal to the length of the string. The null character is not taken into account. That is why in C the `printf()` statement displays the contents of the next character array followed by the character array. In C++, the `cout` also displays the garbage. It is better to take the null character into account while declaring and initializing the string array. Thus, the above character array can be accurately declared as follows:

```
(a) char text[4] = "xyz";
(b) char txt[4] = "abc";
```

Now, you will get the correct output.

Given below are a few programs on one-dimensional arrays

12.3 Program to read 5 integers through keyboard and display them.

```
#include<iostream.h>
#include<conio.h>

int main()
{
   int num[5],i;
   clrscr();
   for(i=0;i<5;i++)
   {
      cout<<"\n Enter any no:";
      cin>>num[i];
   }
      cout<<" Numbers are:";
      for(i=0;i<5;i++)
   {
      cout<<num[i];
   }
return 0;
}
```

OUTPUT

Enter any no:12
Enter any no:34
Enter any no:45
Enter any no:67
Enter any no:89
Numbers are:12 34 45 67 89

12.4 Program to enter 5 numbers through keyboard and display them in ascending order.


```
#include<iostream.h>
#include<conio.h>

int main()
{
   int num[5],i,j,temp;
   clrscr();
   cout<<"\n Enter 5 numbers:";
   for(i=0;i<5;i++)
   {
      cin>>num[i];
```

```
        }
        for(i=0;i<5;i++)
        {
        for(j=i+1;j<5;j++)
        {
        if(num[i]>num[j])
        {
        int temp=num[i];
        num[i]=num[j];
        num[j]=temp;
        }
    }
}
    cout<<"\n Numbers in ascending order are as follows:";
    for(i=0;i<5;i++)
    {
    cout<<" "<<num[i];
    }
return 0;
}
```

OUTPUT

Enter 5 numbers:23 56 78 99 2
Numbers in ascending order are as follows: 2 23 56 78 99

Explanation: For arranging numbers in the ascending order, the numbers are sorted with two for loops, and the same sorted numbers are displayed.

12.4 ACCESSING ARRAY ELEMENTS THROUGH POINTERS

We can quickly and easily access array elements using pointers, as these elements are stored in contiguous memory locations. A pointer variable contains an address, and it is easy to manipulate data with the help of addresses. When a pointer is incremented, the address gets incremented, and we can access the contents of memory locations. This particular method requires less memory; hence, execution is fast.

12.5 Program to display elements of an array using pointer. Display addresses of elements.

```
#include<iostream.h>
#include<conio.h>

int main()
{
    int *p,num[5]={1,2,3,4,5},j;
```

```
    clrscr();
    p=&num[0];
    cout<<"\nNumber Address"<<endl;
    for (j=0;j<5;j++)
    cout<<" "<<*(p+j) <<" "<<unsigned(p+j)<<endl;
    return 0;
}
```

OUTPUT

Number Address
1 65452
2 65454
3 65456
4 65458
5 65460

12.5 ARRAYS OF POINTERS

So far, we have studied arrays of different standard data types, such as arrays of `int`, `float`, and `char`. In the same way, the 'C++' language also supports arrays of pointers. It is nothing but a collection of addresses. Here, we store the addresses of variables for which we have to declare arrays as pointers.

12.6 Write a program to store addresses of different elements of an array using array of pointers.

```
#include<conio.h>
#include<iostream.h>

int main()
{
    int *arrp[3];
    int arr[3]={5,10,15},k;
    for(k=0;k<3;k++)
    arrp[k]=arr+k;
    clrscr();
    cout<<"\n\t Address Element"<<endl;
    for (k=0;k<3;k++)
    {
       cout<<"\t" <<unsigned(arrp[k]);
       cout<<"\t"<<*(arrp[k])<<endl;
    }
return 0;
```

```
}
```

OUTPUT

```
Address  Element
65418    5
65420    10
65422    15
```

Explanation: In the above program, `*arrp[3]` is declared as an array of pointers. Using first `for` loop, the addresses of various elements of array `'arr[]'` are assigned to `'*arrp[]'`. The second `for` loop picks up addresses from `'*arrp[]'` and displays the values present at those locations. Here, each element of `'*arrp[]'` points to the respective element of array `'arr[]'`.

Table 12.2 Arrays of pointers in memory

Element no.	Array of values	Element no.	Array of addresses
`arr[0]`	5	`arrp[0]`	65418
`arr[1]`	10	`arrp[1]`	65420
`arr[2]`	15	`arrp[2]`	65422

12.6 PASSING ARRAY ELEMENTS TO A FUNCTION

We can pass elements to a function by using call-by-value or call-by-reference methods. In the call-by-value method, elements (values) of an array are passed to the function; whereas in the call-by-reference method, addresses of elements are passed to the function.

The following program demonstrates the call-by-value method:

12.7 Program to pass elements of an array to a function by using call by value.

```
#include<iostream.h>
#include<conio.h>
void display(int);

void main()
{
   int num[5]={1,2,3,4,5},i;
   clrscr();
   cout<<"\nElements in the reverse order are as follows:";
   for(i=4;i>=0;i--)
   {
      display(num[i]);
   }
```

```
}
   void display(int x)
{
   cout<<" "<<x;
}
```

OUTPUT

Elements in the reverse order are as follows: 5 4 3 2 1

Explanation: In the above program, individual elements of array num[5] are passed to the function display(`int`), and the same are displayed in the called function.

The following program demonstrates the call-by-reference method:

12.8 Program to pass elements of an array to a function by using call by reference.

```
#include<iostream.h>
#include<conio.h>
void display(int *);

void main()
{
   int num[5]={1,12,3,4,5},i;
   clrscr();
   cout<<"\nElements of the array are as follows:";
   for(i=0;i<5;i++)
   {
      display(&num[i]);
   }
}
void display(int *x)
{
   cout<<" "<<*x;
}
```

OUTPUT

Elements of the array are as follows: 1 2 3 4 5

Explanation: In the above program, addresses of individual array elements are passed to the function display(`int *`). The x contains the address of an array element, and *x is the value stored at that address.

12.7 PASSING COMPLETE ARRAY ELEMENTS TO A FUNCTION

It is also possible to pass the entire elements of an array to a function instead of passing the individual elements of an array. The following program explains the concept:

12.9 Program to pass entire elements of an array to a function.

```
#include<iostream.h>
#include<conio.h>
void show(int *,int);

void main()
{
   int num[5]={1,2,3,4,5},i;
   clrscr();
   cout<<"\nElements of the array are as follows:";
   show(&num[0],5);
}
void show(int *x,int y)
{
   int i;
   for(i=0;i<=y-1;i++)
   {
   cout<<" "<<*x;
   x++;
   }
}
```

OUTPUT

Elements of the array are as follows: 1 2 3 4 5

Explanation: In the above program, the address of the zeroth element, that is, `&num[0]`, is passed to the function `show()`. The `for` loop is used to access the array elements using pointers. The `show()` function is invoked with two arguments. The first argument is the address of the zeroth element, and the second one is the number that represents the total number of elements in the array.

12.8 INITIALIZATION OF ARRAYS USING FUNCTIONS

The programmers always initialize the arrays using statements such as `int d[] = {1,2,3,4,5};` instead of this, the function can also be directly called to initialize the array. The following program illustrates this point.

12.10 Write a program to initialize an array using functions.

```
#include<stdio.h>
#include<conio.h>

main()
{
   int k,c(),d[]={c(),c(),c(),c(),c()};
   clrscr();
```

```
    printf ("\n Array d[] elements are :");
    for (k=0;k<5;k++)
    printf ("%2d",d[k]);
    return (NULL);
}
c()
{
    static int m,n;
    m++;
    printf ("\nEnter Number d[%d] : ",m);
    scanf ("%d",&n);
    return(n);
}
```

OUTPUT

Enter Number d[1] : 4
Enter Number d[2] : 5
Enter Number d[3] : 6
Enter Number d[4] : 7
Enter Number d[5] : 8
Array d[] elements are : 4 5 6 7 8

Explanation: A function can be called in the declaration of an array. In the above program, `d[]` is an integer array, and `c()` is a user-defined function. When called, the function `c()` reads values through the keyboard. The function `c()` is called from an array; that is, the values returned by the function are assigned to the array. The above program will not work in C.

12.9 TWO-DIMENSIONAL ARRAYS

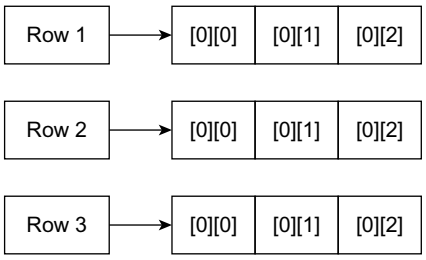

Fig. 12.1 Two-dimensional array

Two-dimensional arrays can be considered a rectangular display of elements with rows and columns, and this is also known as a `matrix`. Consider the following example `int` x[3][3]. The two-dimensional array can be declared as shown in Fig. 12.1.

Table 12.1 Arrangement of two-dimensional array elements

	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

The arrangement of array elements shown in Table 12.1 is only for the sake of understanding. Actually, the elements are stored in continuous memory locations. The two-dimensional array is a collection of two one-dimensional arrays. The meaning of the first argument is in x[3][3]

and means the number of rows; that is, the number of one-dimensional arrays, and the second argument indicates the number of elements. The x[0][0] means the first element of the first row and column. In one row, the row number remains the same but the column number changes. The number of rows and columns is called the `range` of the array. A two-dimensional array clearly shows the difference between logical assumptions and the physical representation of data. The computer memory is linear and any type of array may one, two- or multi-dimensional array it is stored in continuous memory location Fig. 12.2.

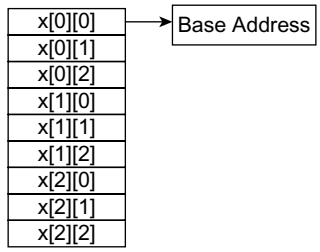

Fig. 12.2 Storage of two-dimensional array

12.11 Program to demonstrate two dimensional array.

```
#include<iostream.h>
#include<conio.h>

int main()
{
   int i,j;
   int a[3][3]={1,2,3,4,5,6,7,8,9};
   clrscr();
   cout<<"\n Array elements and address ";
   cout<<"\n \t Col-0      Col-1     Col-2";
   cout<<"\n \t ======    ======     ======";
   cout<<"\nRow0";
   for (i=0;i<3;i++)
   {
      for (j=0;j<3;j++)
      cout<<"\t      "<< a[i][j];
      if(i==2)
      break;
      cout<<"\nRow"<<i+1;
   }
return 0;
}
```

OUTPUT

Array elements and address

	Col-0	Col-1	Col-2
	======	======	======
Row0	**1**	**2**	**3**
Row1	**4**	**5**	**6**
Row2	**7**	**8**	**9**

Explanation : From the above program's output, we can conclude that the one-dimensional array can be accessed using a single loop. However, for the two-dimensional array, two loops are required for rows and columns. The inner loop helps access the row-wise elements, and the outer loop changes the column number.

12.12 Program to read marks and percentage of students using two dimensional array.


```
#include<iostream.h>
#include<conio.h>

int main()
{
   int student[5][2],i,j;
   clrscr();
   for(i=0;i<5;i++)
   {
   cout<<"\n Enter the Roll no and percentage of the student:";
   cin>>student[i][0]>>student[i][1];
   }
   cout<<"\n Roll_no \t percentage ";
   for(i=0;i<5;i++)
   {
      cout<<"\n";
      cout<<student[i][0]<<"\t"<<(student[i][1]);
   }
return 0;
}
```

OUTPUT

```
Enter the Roll no and percentage of the student:1 89
Enter the Roll no and percentage of the student:2 78
Enter the Roll no and percentage of the student:3 76
Enter the Roll no and percentage of the student:4 56
Enter the Roll no and percentage of the student:5 90
Roll_no    percentage
1             89
2             78
3             76
4             56
5             90
```

Explanation: In the above program, student [5][2] is declared as an array. The array student [5][2] contains 5 rows and 2 columns. Roll number and percentage scored by the students are read through the keyboard and displayed.

12.10 POINTERS AND TWO-DIMENSIONAL ARRAYS

A matrix can represent the two-dimensional elements of an array. In order to display the elements of the two-dimensional array using pointers, it is essential to have **'&'** operator as a prefix with the array name followed by element numbers.

12.13 Write a program to display array elements and their addresses using pointers.

```
#include<iostream.h>
#include<stdio.h>
#include<conio.h>

void main()
{
   clrscr();
   int i,j=1,*p;
   int a[3][3]={{1,2,3},{4,5,6},{7,8,9}};
   cout<<"\n\tElements of an array with their addresses";
   p=&a[0][0];
   cout<<"\n";
   for (i=0;i<9;i++,j++)
   {
   cout<<" "<<*(p)<<" "<<unsigned(p);
   if (j==3)
   {
   cout<<"\n";
   j=0;
   }
   p++;
}
}
```

OUTPUT

Elements of an array with their addresses
1 65506 2 65508 3 65510
4 65512 5 65514 6 65516
7 65518 8 65520 9 65522

Explanation : In the above program, the two-dimensional array is declared and initialized. The base address of the array is assigned to integer pointer `'p'`. While assigning the base address of the two-dimensional array, the `'&'` operator is pre-fixed with the array name followed by the element numbers; otherwise, the compiler shows an error. The statement `p = &a[0][0]` is used in this context. The pointer `'p'` is printed and incremented in the `for` loop till it prints the entire array of elements. The `if` statement splits a line when three elements in each row are printed.

12.11 THREE- OR MULTI-DIMENSIONAL ARRAYS

The ***'C++'*** program helps in creating an array of multi dimensions. The compiler determines the restrictions on it. The syntax of the declaration of multi-dimensional arrays is as follows:

```
Data Type_Arrayname [S1][S2][S3]...[Si];
```

where Si is the size of the i^{th} dimensions.
Three-dimensional arrays can be initialized as follows.

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

A three-dimensional array can be considered an array of arrays of arrays. The outer array contains three elements. The inner array is two dimensional with regard to size [3][3].

12.14 Write a program to explain the working of three-dimensional array.

```
#include<iostream.h>
#include<conio.h>

int main()
{
   int array_3d[3][3][3];
   int a,b,c;
   clrscr();
   for (a=0;a<3;a++)
   for (b=0;b<3;b++)
   for (c=0;c<3;c++)
   array_3d[a][b][c]=a+b+c;
   for (a=0;a<3;a++)
   {
      cout<<"\n";
      for (b=0;b<3;b++)
      {
            for (c=0;c<3;c++)
            cout<<" "<<array_3d[a][b][c];
            cout<<"\n";
      }
   }
return 0;
}
```

OUTPUT

0 1 2
1 2 3
2 3 4

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

Explanation : The three-dimensional array `array_3d` is initialized. The first three `for` loops are used for adding the values of `a, b & c`. Here, initially, `a & b` are zero, and `'c'` varies from 0 to 2. Hence, the addition of `a, b & c` will be 0 1 2. This will be printed in the first row. The second output row is one in which `a = 0, b = 1,` and `c` varies from 0 to 2. Thus, the output of the second row will be 1 2 3. In this way, the values of `a, b & c` are changed, and a total of 27 iterations are carried out.

12.12 ARRAYS OF CLASSES

As we know, an array is a collection of similar data types. In the same way, we can also define an array of classes. In such a type of an array, every element is of class type. An array of class objects can be declared as follows.

```
class stud
{
public:
char name[12];      // class declaration
int rollno;
char grade[2];
};
class stud st[3]; // declaration of array of class objects
```

In the above example, `st[3]` is an array of three elements containing three objects of `class stud`. Each element of `st[3]` has its own set class of member variables; that is, `char name[12], int rollno,` and `char grade[2]`. A program is explained as given below.

12.15 Write a program to display names, roll numbers, and grades of 3 students who have appeared for the examination. Declare the class of name, roll numbers, and grade. Create an array of class objects. Read and display the contents of the array.


```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>

void main()
{
   int k=0;
   class stud
   {
```

```
      public:
      char name[12];
      int rollno;
   char grade[2];
};
class stud st[3];
while (k<3)
{  clrscr();
   gotoxy(2,4);
   cout<<"Name:";
   gotoxy(17,4);
   cin>>st[k].name;
   gotoxy(2,5);
   cout<<"Roll No.:";
   gotoxy(17,5);
   cin>>st[k].rollno;
   gotoxy(2,6);
   cout<<"Grade:";
   gotoxy(17,6);
   cin>>st[k].grade;
   st[k].grade[1]='\0';
   puts ("press any key..");
   getch();
   k++;
}
   k=0;
   clrscr();
   cout<<"\nName\tRollno Grade\n";
   while (k<3)
   {
   cout<<st[k].name<<"\t"<<st[k].rollno<<" \t"<<st[k].
         grade<<"\n";
   k++;
}
}
```

OUTPUT

Name	**Rollno**	**Grade**
Balaji	**50**	**A**
Manoj	**51**	**B**
Sanjay	**55**	**C**

Explanation: In the above program, class `stud` is declared. Its members are `char name[12]`, `int rollno`, and `char grade[2]`, and all are public. The array `st[3]` of `class stud` is declared. The first `while` loop and `cin()` statements within it are used for repetitive data reading. The second `while` loop and `printf()` statements within it display the contents of the array. In the above program, all the member variables are public. If the members are private, the above program will not work. Next, we need to declare member functions to access the data. The program given below explains the reading of private data using member functions.

12.16 Write a program to display names, roll numbers, and grades of 3 students who have appeared for the examination. Declare the class of name, roll numbers, and grade private. Create an array of class objects. Read and display the contents of the array using member functions.

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>
void put()
{
cout<<name<<"\t"<<rollno<<"\t"<<grade<<"\n";
}
};
class stud
{
   private :
   char name[12];
   int rollno;
   char grade[2];
   public :
void get()
{  clrscr();
   gotoxy(2,4);
   cout<<"Name      : ";
   gotoxy(17,4);
   cin>>name;
   gotoxy(2,5);
   cout<<"Roll No. : ";
   gotoxy(17,5);
   cin>>rollno;
   gotoxy(2,6);
   cout<<"Grade     :";
   gotoxy(17,6);
   cin>>grade;
   grade[1]='\0';
```

```
    puts ("press any key..");
    getch();
}
int main()
{
int m=0;
class stud st[3];
while (m<3)
{
    st[m].get();
    m++;
    }
    cout<<"\nName\tRollno Grade\n";
    m=0;
    while (m<3)
    {
    st[m].put();
    m++;
    }
return 0;
}
```

OUTPUT

Name Rollno Grade
Balaji 50 A
Manoj 51 B
Sanjay 55 C

Explanation: The above program is similar to the previous one. Here, the class members are private. It is not possible to access the private members directly. Using member functions `get()` and `put()`, data are read and displayed.

SUMMARY

(1) An array is a collection of similar data types that are stored in different memory locations.
(2) The array elements are stored in continuous memory locations. The amount of storage required for holding the elements of an array depends on its type and size.
(3) The declaration and initialization of one-, two-, and multi-dimensional arrays are studied in this chapter with programming examples.
(4) How can array elements be accessed through pointers? The answer to this question is supported by theory and programs.
(5) Passing individual array elements to a function and passing entire elements to a function are studied in this chapter.

(6) Pointers and two-dimensional arrays are dealt with in this chapter.

(7) Arrays and classes are also explained.

EXERCISES

(A) Answer the following questions

(1) What are arrays? How are the elements of an array stored?
(2) Explain two-dimensional arrays.
(3) Explain multi-dimensional arrays.
(4) The array name contains the base address of the array. Can we change the base address of the array?
(5) What is the relationship between array name and element number? How are elements referred to by using the base address?
(6) Can we store values and addresses in the same array? Explain.
(7) Mention the differences between character array and integer array.
(8) Explain the accessing of array elements by passing value to a function with a programming example.
(9) Explain the meaning of `call by reference` with a programming example.
(10) What do you mean by initialization of array with function?

(B) Answer the following by selecting the appropriate option

(1) What will happen if you assign values in a few locations of an array?
   (a) Rest of the elements will be set to 0
   (b) Compiler error message will be displayed
   (c) Possible system crash
(2) When an array is passed to function, what gets passed in reality?
   (a) Address of the array
   (b) Element numbers
   (c) Values of the array
(3) Which is the correct statement to declare an array?
   (a) `int x[];`
   (b) `int x[5];`
   (c) `int x{5}'`
(4) The array name itself is a pointer to
   (a) 0^{th} element
   (b) 1^{st} element
   (c) last element
(5) The array name itself is a
   (a) constant pointer objet
   (b) pointer
   (c) address
   (d) none of the above
(6) `int x[5],*p;` and p = x; then, following which one operation is wrong?
   (a) x++;
   (b) p++;
   (c) p = x;
(7) `int x[3];` the base address of x is 65460, and the elements are stored at locations
   (a) 65460, 65462, 65464
   (b) 65460, 65461, 65462
   (c) 65460, 65464, 65468
(8) `int j[4]` the `sizeof(j)` and `sizeof(int )` will display the value
   (a) 8, 2
   (b) 2, 8
   (c) 2, 2

(C) Attempt the following programs

(1) Write a program to display all the elements of a character array using pointers.
(2) Write a program to read the elements of two-dimensional arrays and display them with and without pointers.
(3) Write a program to read five elements from an array. Perform addition and multiplication of these elements.
(4) Write a program to read a number containing three digits. Perform the square

roots of each digit. For example, the number is 149. Output should be the square root of each digit, that is, 1 2 3.

(5) Write a program to read the numbers of any lengths. Count the odd numbers and display them in ascending order.

(6) Write a program to find the total marks of three subjects Physics, Chemistry, and Math of three students. Display their ids and marks with the declaration and initialization of arrays.

(7) Write a program to initialize two arrays having equal elements. Compute addition and subtraction of the respective elements of both these arrays and display them.

(8) Write a program to initialize and read 26 six alphabets in upper case and their ASCII values in reverse (Z to A).

(9) Write a program to enter a string in lower as well as in upper case. Convert lower case to upper case and vice versa and display them in reverse order.

(10) Read the marks of five subjects obtained by five students in an examination. Display the top two student codes and their marks.

(11) Initialize two arrays and merge them. Display the elements in descending order.

(12) Write a program to perform the transpose of the matrix.

(13) Write a program to check whether the given string is a palindrome.

(14) Write a program to sort the numbers using the bubble-sort method.

(15) Write a program to sort the numbers using the selection-sort method.

(16) Write a program to search for the target element in an array and its position from the last element.

(17) Write a program to merge two arrays.

(18) Write a program to compute the sum of even and odd numbers.

(19) Write a program on the multiplication table of the entered number.