UNIT 2 ARRAYS

Structure **Page Nos.**

2.0 INTRODUCTION

This unit introduces a data structure called Arrays. The simplest form of array is a one-dimensional array that may be defined as a finite ordered set of homogeneous elements, which is stored in contiguous memory locations. For example, an array may contain all integers or all characters or any other data type, but may not contain a mix of data types.

The general form for declaring a single dimensional array is:

data_type array_name[expression];

where data_type represents data type of the array. That is, integer, char, float etc. array_name is the name of array and expression which indicates the number of elements in the array.

For example, consider the following C declaration:

```
int   a[100];
```

It declares an array of 100 integers.

The amount of storage required to hold an array is directly related to its type and size. For a single dimension array, the total size in bytes required for the array is computed as shown below.

Memory required (in bytes) = size of (data type) X length of array

The first array index value is referred to as its lower bound and in C it is always 0 and the maximum index value is called its upper bound. The number of elements in the array, called its range is given by upper bound-lower bound.

We store values in the arrays during program execution. Let us now see the process of initializing an array while declaring it.

```
int a[4] = {34,60,93,2};
int b[] = {2,3,4,5};
float c[] = {-4,6,81,– 60};
```

We conclude the following facts from these examples:

(i) If the array is initialized at the time of declaration, then the dimension of the array is optional.
(ii) Till the array elements are not given any specific values, they contain garbage values.

2.1 OBJECTIVES

After going through this unit, you will be able to:

- use Arrays as a proper data structure in programs;
- know the advantages and disadvantages of Arrays;
- use multidimensional arrays, and
- know the representation of Arrays in memory.

2.2 ARRAYS AND POINTERS

C compiler does not check the bounds of arrays. It is your job to do the necessary work for checking boundaries wherever needed.

One of the most common arrays is a string, which is simply an array of characters terminated by a null character. The value of the null character is zero. A string constant is a one-dimensional array of characters terminated by a null character(\0).

For example, consider the following:

```
char message[ ]= {'e', 'x', 'a', 'm', 'p', 'l','e','\0'};
```

Also, consider the following string which is stored in an array:

"sentence\n"

Figure 2.1 shows the way a character array is stored in memory. Each character in the array occupies one byte of memory and the last character is always '\0'. Note that '\0' and '0' are not the same. The elements of the character array are stored in contiguous memory locations.

s	**e**	**n**	**t**	**e**	**n**	**c**	**e**	**\n**	**\0**

Figure 2.1: String in Memory

C concedes a fact that the user would use strings very often and hence provides a short cut for initialization of strings.

For example, the string used above can also be initialized as

```
char name[ ] = "sentence\n";
```

Note that, in this declaration '\0' is not necessary. C inserts the null character automatically.

Multidimensional arrays are defined in the same manner as one-dimensional arrays, except that a separate pair of square brackets is required for each subscript. Thus a two-dimensional array will require two pairs of square brackets, a three-dimensional array will require three pairs of square brackets and so on.

The format of declaration of a multidimensional array in C is given below:

```
data_type array_name [expr 1] [expr 2] …. [expr n];
```

where data_type is the type of array such as int, char etc., array_name is the name of array and expr 1, expr 2, ….expr n are positive valued integer expressions.

The schematic of a two-dimensional array of size 3 × 5 is shown in *Figure 2.2.*

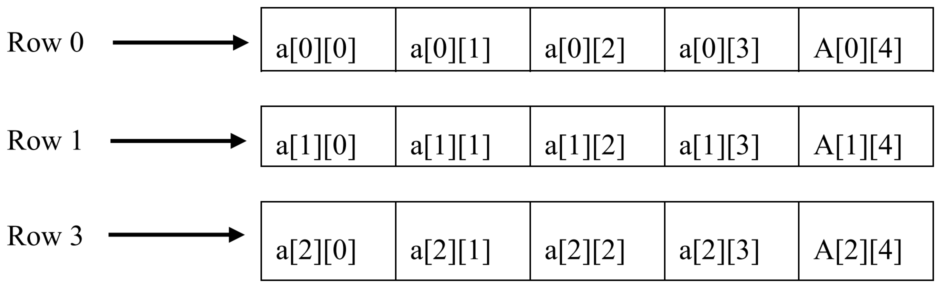

Figure 2.2: Schematic of a Two-Dimensional Array

In the case of a two-dimensional array, the following formula yields the number of bytes of memory needed to hold it:

bytes = size of 1st index × size of 2nd index × size of (base type)

The pointers and arrays are closely related. As you know, an array name without an index is a pointer to the first element in the array.

Consider the following array:

```
char p[10];
```

p and &p[0] are identical because the address of the first element of an array is the same as the address of the array. So, an array name without an index generates a pointer. Conversely a pointer can be indexed as if it were declared to be an array.

For example, consider the following program fragment:

```
    int *x, a [10];
    x = a;
    x[5] = 100;
  * (x+5) = 100;
```

Both assignment statements place the value 100 in the sixth element of a. Furthermore the (0,4) element of a two-dimensional array may be referenced in the following two ways: either by array indexing a[0][4], or by the pointer *((int *) a+4).

In general, for any two-dimensional array a[j][k] is equivalent to:

*((base type *)a + (j * rowlength)*k)

2.3 SPARSE MATRICES

Matrices with good number of zero entries are called sparse matrices.

Consider the following matrices of *Figure 2.3.*

$$\begin{pmatrix} 4 & & & & \\ 3 & -5 & & & \\ 1 & 0 & 6 & & \\ -7 & 8 & -1 & 3 & \\ 5 & -2 & 0 & 2 & -8 \end{pmatrix} \qquad \begin{pmatrix} 5 & -3 & & & & & \\ 1 & 4 & 3 & & & & \\ & 9 & -3 & 6 & & & \\ & & 2 & 4 & -7 & & \\ & & & 3 & -1 & 0 & \\ & & & & 6 & -5 & 8 \\ & & & & & 3 & -1 \end{pmatrix}$$

(a) **(b)**

Figure 2.3: (a) Triangular Matrix (b) Tridiagonal Matrix

A triangular matrix is a square matrix in which all the elements either above or below the main diagonal are zero. Triangular matrices are sparse matrices. A tridiagonal matrix is a square matrix in which all the elements except for the main diagonal, diagonals on the immediate upper and lower side are zeroes. Tridiagonal matrices are also sparse matrices.

Let us consider a sparse matrix from storage point of view. Suppose that the entire sparse matrix is stored. Then, a considerable amount of memory which stores the matrix consists of zeroes. This is nothing but wastage of memory. In real life applications, such wastage may count to megabytes. So, an efficient method of storing sparse matrices has to be looked into.

Figure 2.4 shows a sparse matrix of order 7 × 6.

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

Figure 2.4: Representation of a sparse matrix of order 7 × 6

A common way of representing non zero elements of a sparse matrix is the 3-tuple form. The first row of sparse matrix always specifies the number of rows, number of columns and number of non zero elements in the matrix. The number 7 represents the total number of rows sparse matrix. Similarly, the number 6 represents the total number of columns in the matrix. The number 8 represents the total number of non zero elements in the matrix. Each non zero element is stored from the second row, with the 1st and 2nd elements of the row, indicating the row number and column number respectively in which the element is present in the original matrix. The 3rd element in this row stores the actual value of the non zero element. For example, the 3- tuple representation of the matrix of *Figure 2.4* is shown in *Figure 2.5*.

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

Figure 2.5: 3-tuple representation of Figure 2.4

The following program 1.1 accepts a matrix as input, which is sparse and prints the corresponding 3-tuple representations.

Program 1.1

/* The program accepts a matrix as input and prints the 3-tuple representation of it*/

```
#include<stdio.h>

void main()
{
        int a[5][5],rows,columns,i,j;

        printf("enter the order of the matrix. The order should be less than 5 × 5:\n");
        scanf("%d %d",&rows,&columns);
        printf("Enter the elements of the matrix:\n");

        for(i=0;i<rows;i++)
          for(j=0;j<columns;j++)

          { scanf("%d",&a[i][j]);
          }
           printf("The 3-tuple representation of the matrix is:\n");

        for(i=0;i<rows;i++)
                for(j=0;j<columns;j++)
                {
                        if (a[i][j]!=0)
                        {
                                printf("%d    %d       %d\n", (i+1),(j+1),a[i][j]);
                        }
                }
}
```

Output:

```
enter the order of the matrix. The order should be less than 5 × 5:
3 3
Enter the elements of the matrix:
1 2 3
0 1 0
0 0 4
The 3-tuple representation of the matrix is:
1   1     1
1   2     2
1   3     3
2   2     1
3   3     4
```

The program initially prompted for the order of the input matrix with a warning that the order should not be greater than 5 × 5. After accepting the order, it prompts for the elements of the matrix. After accepting the matrix, it checks each element of the matrix for a non zero. If the element is non zero, then it prints the row number and column number of that element along with its value.

☞ Check Your Progress 1

1) If the array is _______ at the time of declaration, then the dimension of the array is optional.

2) A sparse matrix is a matrix which is having good number of _____ elements.

3) At maximum, an array can be a two-dimensional array. True/False

2.4 POLYNOMIALS

Polynomials like $5x^4 + 2x^3 + 7x^2 + 10x - 8$ can be represented using arrays. Arithmetic operations like addition and multiplication of polynomials are common and most often, we need to write a program to implement these operations.

The simplest way to represent a polynomial of degree '*n*' is to store the coefficient of (*n+1*) terms of the polynomial in an array. To achieve this, each element of the array should consist of two values, namely, coefficient and exponent. While maintaining the polynomial, it is assumed that the exponent of each successive term is less than that of the previous term. Once we build an array to represent a polynomial, we can use such an array to perform common polynomial operations like addition and multiplication.

Program 1.2 accepts two polynomials as input and adds them.

Program 1.2

/* The program accepts two polynomials as input and prints the resultant polynomial due to the addition of input polynomials*/

```
#include<stdio.h>

void main()
{
        int poly1[6][2],poly2[6][2],term1,term2,match,proceed,i,j;

        printf("Enter the number of terms in the first polynomial. They should be less
than 6:\n");
        scanf("%d",&term1);
        printf("Enter the number of terms in the second polynomial. They should be
less than 6:\n");
        scanf("%d",&term2);
        printf("Enter the coefficient and exponent of each term of the first
polynomial:\n");
        for(i=0;i<term1;i++)
        {scanf("%d %d",&poly1[i][0],&poly1[i][1]);
        }

  printf("Enter the coefficient and exponent of each term of the second
polynomial:\n");
  for(i=0;i<term2;i++)
        {scanf("%d %d",&poly2[i][0],&poly2[i][1]);
```

```
          }
   printf("The resultant polynomial due to the addition of the input two
polynomials:\n");

          for(i=0;i<term1;i++)
          {
                    match=0;
                    for(j=0;j<term2;j++)

                    { if (match==0)

                              if(poly1[i][1]==poly2[j][1])
                              { printf("%d    %d\n",(poly1[i][0]+poly2[j][0]), poly1[i][1]);
                                match=1;

                              }
                    }
          }

for(i=0;i<term1;i++)
{  proceed=1;

                    for(j=0;j<term2;j++)
                    {  if(proceed==1)
                              if(poly1[i][1]!=poly2[j][1])
                               proceed=1;
                              else
                                        proceed=0;

                              }
                    if (proceed==1)
                              printf("%d %d\n",poly1[i][0],poly1[i][1]);

          }

for(i=0;i<term2;i++)
{  proceed=1;

                    for(j=0;j<term1;j++)
                    {  if(proceed==1)
                              if(poly2[i][1]!=poly1[j][1])
                               proceed=1;
                              else
                                        proceed=0;

                              }
                    if (proceed==1)
                              printf("%d %d",poly2[i][0],poly2[i][1]);
                    }

}
```

Output:

```
Enter the number of terms in the first polynomial.They should be less than 6 : 5.
Enter the number of terms in the second polynomial.They should be less than 6 : 4.
Enter the coefficient and exponent of each term of the first polynomial:
1 2
2 4
3 6
```

```
1 8
5 7

Enter the coefficient and exponent of each term of the second polynomial:
5 2
6 9
3 6
5 7

The resultant polynomial due to the addition of the input two polynomials:
6 2
6 6
10 7
2 4
1 8
6 9
```

The program initially prompted for the number of terms of the two polynomials. Then, it prompted for the entry of the terms of the two polynomials one after another. Initially, it adds the coefficients of the corresponding terms of both the polynomials whose exponents are the same. Then, it prints the terms of the first polynomial who does not have corresponding terms in the second polynomial with the same exponent. Finally, it prints the terms of the second polynomial who does not have corresponding terms in the first polynomial.

2.5 REPRESENTATION OF ARRAYS

It is not uncommon to find a large number of programs which process the elements of an array in sequence. But, does it mean that the elements of an array are also stored in sequence in memory. The answer depends on the operating system under which the program is running. However, the elements of an array are stored in sequence to the extent possible. If they are being stored in sequence, then how are they sequenced. Is it that the elements are stored row wise or column wise? Again, it depends on the operating system. The former is called row major order and the later is called column major order.

2.5.1 Row Major Representation

The first method of representing a two-dimensional array in memory is the row major representation. Under this representation, the first row of the array occupies the first set of the memory location reserved for the array, the second row occupies the next set, and so forth.

The schematic of row major representation of an Array is shown in *Figure 2.6.*
Let us consider the following two-dimensional array:

$$\begin{array}{cccc} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{array}$$

To make its equivalent row major representation, we perform the following process:

Move the elements of the second row starting from the first element to the memory location adjacent to the last element of the first row. When this step is applied to all the rows except for the first row, you have a single row of elements. This is the Row major representation.

By application of above mentioned process, we get {a, b, c, d, e, f, g, h, i, j, k, l }

Row 0	Row 1	Row 2	…..	Row i		

Figure 2.6: Schematic of a Row major representation of an Array

2.5.2 Column Major Representation

The second method of representing a two-dimensional array in memory is the column major representation. Under this representation, the first column of the array occupies the first set of the memory locations reserved for the array. The second column occupies the next set and so forth. The schematic of a column major representation is shown in *Figure 2.7.*

Consider the following two-dimensional array:

$$\begin{matrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{matrix}$$

To make its equivalent column major representation, we perform the following process:

Transpose the elements of the array. Then, the representation will be same as that of the row major representation.

By application of above mentioned process, we get {a, e, i, b, f, j, c, g, k, d, h, i}

Col 0	Col 1	Col 2	…..	Col i		

Figure 2.7: Schematic of a Column major representation of an Array

☞ Check Your Progress 2

1) An array can be stored either__________or _________.

2) In __________, the elements of array are stored row wise.

3) In __________, the elements of array are stored column wise.

2.6 APPLICATIONS

Arrays are simple, but reliable to use in more situations than you can count. Arrays are used in those problems when the number of items to be solved is fixed. They are easy to traverse, search and sort. It is very easy to manipulate an array rather than other subsequent data structures. Arrays are used in those situations where in the size of array can be established before hand. Also, they are used in situations where the insertions and deletions are minimal or not present. Insertion and deletion operations will lead to wastage of memory or will increase the time complexity of the program due to the reshuffling of elements.

2.7 SUMMARY

In this unit, we discussed the data structure **arrays** from the application point of view and representation point of view. Two applications namely representation of a sparse matrix in a 3-tuple form and addition of two polynomials are given in the form of programs. The format for declaration and utility of both single and two-dimensional arrays are covered. Finally, the most important issue of representation was discussed. As part of it, row major and column major orders are discussed.

2.8 SOLUTIONS / ANSWERS

Check Your Progress 1

1) Initialized
2) Zero
3) False

Check Your Progress 2

1) Row wise, column wise
2) Row major representation
3) Column major representation

2.9 FURTHER READINGS

Reference Books

1. *Data Structures using C and C++*, Yedidyah Langsam, Moshe J.Augenstein, Aaron M Tanenbaum, Second Edition, PHI Publications.
2. *Data Structures, Seymour Lipscutz, Schaum's outline series*, McGraw Hill

Reference Websites

http://www.webopedia.com