4.1 Representations of Two-Dimensional Arrays using One-Dimensional Arrays

It is time to return to the problem of how to represent higher-dimensional arrays in terms of one-dimensional arrays. For simplicity, only two-dimensional arrays are discussed, but the techniques can be readily extended for higher dimensions. Two techniques will be studied: rowwise and columnwise representation.

What exactly is meant by representing a two-dimensional array \( a \) in terms of a one-dimensional array (say, \( \text{data} \)), and why is it done? The idea is that the problem to be solved may involve the use of \( a \), but the language being used may not provide two-dimensional arrays, or the programmer may choose not to use them for some other reason. Instead, the entries of \( a \) are stored in \( \text{data} \). This must be done so that whenever reference is to be made to the entry in position \((i, j)\) of \( a \) in the program, the corresponding position of \( \text{data} \) in which it is stored is referenced instead.

4.1.1 Rowwise and Columnwise Representation

Suppose \( a \) has \( r \) rows and \( c \) columns, with row index range \( 0 \ldots (r - 1) \) and column index range \( 0 \ldots (c - 1) \). A **rowwise representation** for \( a \) uses a \( \text{data} \) array of length \( r \times c \) and stores the first row of \( a \) in the first \( c \) consecutive entries of \( \text{data} \), the second row in the next \( c \) entries, and so on.

Rowwise representation in arrays is logically the same as viewing each row as a record of length \( c \). Thus the offset of the entry in position \((i, j)\) of \( a \) is \( i \times c + j \) in \( \text{data} \). This is because rows \( 0, 1, \ldots, i - 1 \) precede the \( i \)th row in \( \text{data} \), and take up \( i \times c \) entries. The \( j \)th position of the \( i \)th row is offset an additional \( j \) entries.

For the array \( a \) shown in Figure 4.1, with \( c = 5 \), the offset of \( a[2][3] \) is \( 2 \times 5 + 3 \), or 13, and the third row of \( a \) starts at \( \text{data}[10] \). \( a[2][3] \) is represented by \( \text{data}[13] \). In C the offset and index are the same, since index values start at zero.
Obviously, this representation in data allows a to be traversed, as well as allowing its \((i, j)\)th entry to be selected.

The columnwise representation of a two-dimensional array is similar, except that the columns rather than the rows of \(a\) are stored sequentially in data. The offset for the entry in position \((i, j)\) with the columnwise representation is \(j \times r + i\).

### 4.2 Three-dimensional arrays

Generally, an array having one dimension is called 1D array, array having two dimensions called 2D array and so on. So in C programming an array can have two or three or four or even ten or more dimensions. More dimensions in an array means more data it can hold and of course more difficulties to manage and understand these arrays. A multidimensional array has following syntax:

**Syntax:**

```
type array_name[d1][d2][d3][d4]........[dn];
```

Where \(dn\) is the size of last dimension.

**Example:**

```c
int table[5][5][20];
float arr[5][6][5][6][5];
```

In our example array “table” is a 3D (A 3D array is an array of arrays of arrays.) array which can hold 500 integer type elements. And array “arr” is a 5D array which can hold 4500 floating-point elements. Can see the power of array over variable? When it comes to hold multiple values in a C programming, we
need to declare several variables (for example to store 150 integers) but in case of array, a single array can hold thousands of values (depending on compiler, array type etc).

Note: To make this multidimensional array example simple, we will discuss 3D array for the sake of simplicity. Once you grab the logic how 3D array works then you can handle 4D array or any multidimensional array easily.

4.2.1 Declaration and Initialization 3D Array

Before we move to serious programming let's have a look of 3D array. A 3D array can be assumed as an array of arrays of arrays, it is array (collection) of 2D arrays and as you know 2D array itself is array of 1D array. It sounds a bit confusing but don't worry as you will lead your learning on multidimensional array, you will grasp all logic and concept. A diagram can help you to understand this.

We can initialize a 3D array at the compile time as we initialize any other variable or array, by default an un-initialized 3D array contains garbage value. Let's see a complete example on how we can work with a 3D array.

Example1: 3D array

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

Void main()
{
    int numbers[2][1][3];
    int i,j,k;
    clrscr();
    printf(”Enter table values
”);
    for(i=0;i<2;i++)
        {
```
```c
#include<stdio.h>
#include<conio.h>

void main()
{
    int i, j, k;
    int arr[3][3][3]=
    {
        {11, 12, 13},
    }
    printf("Table %d:
",i+1);
    for(j=0;j<1;j++)
    {
        for(k=0;k<3;k++)
            scanf("%d",&numbers[i][j][k]);
    }
    for(i=0;i<2;i++)
    {
        printf("Table %d:
",i+1);
        for(j=0;j<1;j++)
        {
            for(k=0;k<3;k++)
                printf("%d	",numbers[i][j][k]);
            printf("\n")
        }
    }
    getch();
}
```

**Output**

```
Enter table values
Table 1:
1
2
3
Table 2:
4
5
6
Table 1:
1  2  3
Table 2:
4  5  6
```

**Example2: Declaration and Initialization 3D Array**

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

void main()
{
    int i, j, k;
    int arr[3][3][3]=
    {
        {11, 12, 13},
    }
    printf("Table %d:
",i+1);
    for(j=0;j<1;j++)
    {
        for(k=0;k<3;k++)
            scanf("%d",&numbers[i][j][k]);
    }
    for(i=0;i<2;i++)
    {
        printf("Table %d:
",i+1);
        for(j=0;j<1;j++)
        {
            for(k=0;k<3;k++)
                printf("%d	",numbers[i][j][k]);
            printf("\n")
        }
    }
    getch();
}
```
So in the above example we have declared multidimensional array and named this integer array as “arr” which can hold 3x3x3 (27 integers) elements. We have also initialized multidimensional array with some integer values.

As we said earlier that a 3D array is array of 2D array therefore I have divided elements accordingly so that you can get 3D array better and understand it easily. See the C code sample above, line no. 9-13, 14-
18 and 19-23, each block is a 2D array and collectively from line no. 2-24 makes a 3D array. You can also assign values to this multidimensional array in other way like this.

```c
int arr[3][3][3] = {11, 12, 13, 14, 15, 16, 17, 18, 19, 22, 23, 24, 25, 26, 27, 28, 29, 31, 32, 33, 34, 35, 36, 37, 38, 39};
```

This kind of C multidimensional array (3D array) declaration is quite confusing for new C programmers; you cannot guess location of array element by just looking at the declaration. But look at the above multidimensional array example where you can get a clear idea about each element location. For example, consider 3D array as a collection of tables, to access or store any element in a 3D array you need to know first table number then row number and lastly column number. For instance you need to access value 25 from above 3D array. So, first check the table (among 3 tables which table has the value), once you find the table number now check which row of that table has the value again if you get the row no then check column number and you will get the value. So applying above logic, 25 located in table no. 1 row no. 1 and column no. 1, hence the address is arr[1][1][1]. Print this address and you will get the output.

So the conceptual syntax for 3D array stands like this.

```c
data_type array_name[table][row][column];
```

If you want to store values in any 3D array then first point to table number, row number and lastly to column number.

```c
arr[0][1][2] = 32;
arr[1][0][1] = 49;
```

Above code is for assigning values at particular location of an array but if you want to store value in continuous location of array then you should use loop. Here is an example using for loop.

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

void main()
{
    int i, j, k, x=1;
    int arr[3][3][3];
    clrscr();
    printf(":::3D Array Elements:::\n\n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            for(k=0;k<3;k++)
            {
                arr[i][j][k] = x;
                printf("%d\t",arr[i][j][k]);
                x++;
            }
            printf("\n");
        }
    }
}
```
} printf("\n");
} 
getch();