Multi-Dimensional arrays……

An array having more then one dimension is known as multi dimensional arrays. Two dimensional array is also an example of multi dimensional array. One can specify as many dimensions as required. There is no limit on specifying number of dimensions. All dimensions may or may not be the same. The syntax for creating multi dimensions arrays is –

Datatype arrayname[size1][size2]size3].....

Two-Dimensional Arrays.....

An array is a collection of similar elements that is all elements should have same data type. In 2-D arrays, two dimensions that is two sizes are given. First size represents number of rows where as second size represented no of columns. Both the size is may may not be the same. The syntax for creating two dimension is →

Datatype arrayname[size1]size2];

A 2-D array can be think as a table which will have ‘i’ number of rows and ‘j’ number of columns. A 2-D array “a” which contains 3 rows and 4 columns can be shown as below:

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Thus, every element in array “a” is identified by an element name of the form **a[i][j]**, where a is the name of the array, and i and j are the subscripts that uniquely identify each element in a.

Initialization of 2-D arrays……

Initialization means giving values to the arrays. There are two ways of giving values to the arrays i.e.

1- Static initialization

2- Dynamic initialization

Static initialization…….

In static or compile time we give values to arrays while creating the program. Multidimensional arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row have 4 columns.

```
int a[3][4] = {
{0, 1, 2, 3} , /* initializers for row indexed by 0 */
{4, 5, 6, 7} , /* initializers for row indexed by 1 */
{8, 9, 10, 11} /* initializers for row indexed by 2 */
};
```

Example of static initialization

```
int main()
{
    int i,j;
    int a[2][2] = {10,20,30,40};  // declaring and Initializing
  array
    for (i=0;i<2;i++)
   {
   for (j=0;j<2;j++)
     {
    printf("Elements of array is %d\n",a[i][j]); // Accessing
  variables
       }
    }
  }
```

Dynamic initialization…..

Indivisual elements of array can be initialised using the following syntax :-

```
#include<stdio.h>
int main()
{
    int i,j;
    int a[2][2];      // declaring
   for (i=0;i<2;i++)
  {
   for (j=0;j<2;j++)
     {
    scanf("%d",&a[i][j]);     // initializing values to the arrays
       }
    }
  }
```

Accessing 2-D arrays……

An element in 2-dimensional array is accessed by using the subscripts i.e. row index and column index of the array.Elements of 2-d array can be accessed as follow….

```
a[0][0]=10; //assigns 10 to element at 1 row & 1 column
a[0][1]=18; //assigns 10 to element at 1 row & 2 column
a[1][0]=16; //assigns 10 to element at 2 row & 1 column
a[1][1]=10; //assigns 10 to element at 2 row & 2 column
a[2][0]=10; //assigns 10 to element at 3 row & 1 column
```

```
main()
{  int a[3][3],I,j;
printf("Enter the elements in array");
for(i=0; i<=2;i++)
{
for(j=0; j<=2; j++)
{
 scanf("%d",&a[i][j]);
}
}
printf("The elements of array are ");
for(i=0; i<=2;i++)
{
for(j=0; j<=2; j++)
{
 printf("%d",a[i][j]);
}
printf("\n");
}
getch();
}
```

Implementation of 2-d arrays……..

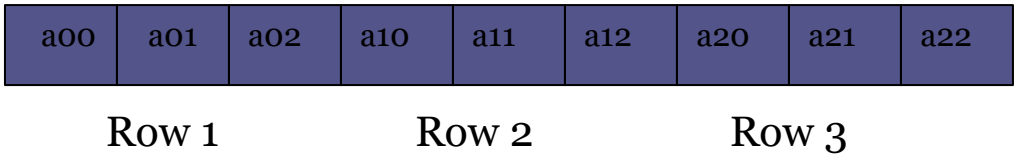
Implementation is the realization of an application, or execution of a plan, idea, model, design etc. It is the process of moving an idea from concept to reality. A two dimensional array can be implemented in a programming language in two ways :-

1-Row major implementation

2-Column major implementation

Row-major implementation……

Row major implementation is linearization technique in which elements of arrays is read from keyboard row-wise i.e. the complete first row is stored ,then complete second row is stored and so on. e.g. an array a[3][3] is stored in memory as shown below :-



We can easily understand the arrangement of array as arranging them in a matrix form like :-

a =

a 00	a 01	a 02
a10	a 11	a 12
a 20	a21	a 22

row1
row2
row3

above figure shows the actual physical storage of elements of the array, whereas the matrix form is logical representation of array(and not the actual way in which 2-D array is stored).

Address of element in row major implementation.....

The computer doesn't keep the track of all elements of the array, rather, it keeps a base address(i.e. the address of first element in array), and calculates the address of required element when needed. It calculates this in row major implementation by the following :-

Address of element : $a[i][j]=B+W(n(i-L_1)+(j-L_2))$

B*=base address, *W*=size of array's datatype, *n*=number of columns(i.e.** U_2-L_2), L_1 ***is lower bound of row & L_2 ***is lower bound of column***

Column major implementation……

- In column major implementation memory allocation is done column by column i.e. all the element of first column is stored, then all the elements of second column is stored , and so on. For example an array a[3][3] is stored in the memory as shown below:-

- column1 column2 column3

Address of element in column major implementation……

In column major implementation address of an element a[i][j] is given by the following relation:-

Address of element : $a[i][j]=B+W(m(j-L_2)+(i-L_1))$

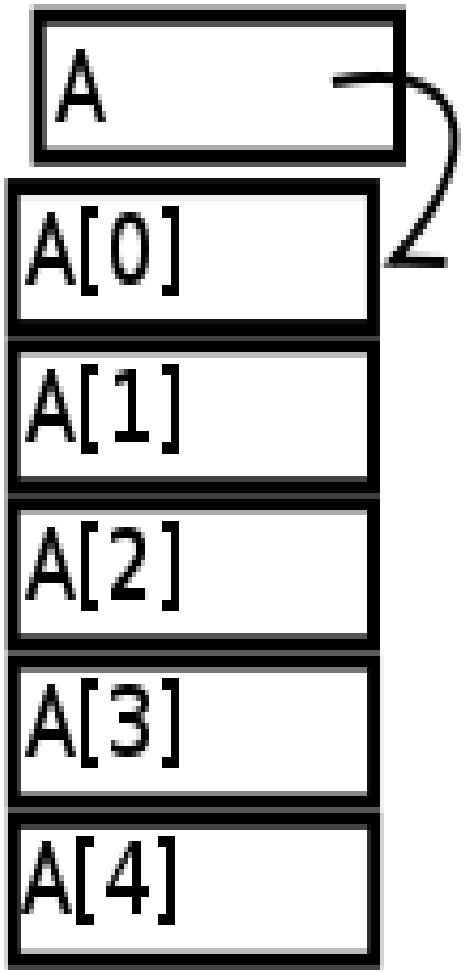
Where m is the number of rows(i.e. ***(i.e.*** U_1-L_1)

POINTERS AND ARRAYS

POINTERS AND ARRAYS

- An array is a collection of homogeneous/similar type of elements stored in adjacent memory locations. Array and pointers have a close link, in fact array in itself acts like a pointer, as one element in the array is stored adjacent to the previous. Therefore while displaying the element of array, the next adjacent address is automatically called when its previous element is displayed. But in case of using pointers with arrays , the element can be stored in arrays at different locations and can be called from that location when needed.

- Before starting the use of pointers with arrays ,two facts must be kept in mind:
- Array element are always stored in continuous locations.
- The incrementation or decrementation of pointers leads to incrementation or decrementation of address based on the type of pointer defined.



Array Access	Pointer Equivalent
arr[0]	*arr
arr[2]	*(arr + 2)
arr[n]	*(arr + n)

Pointer and One-Dimensional array

- Pointer is a variable that can store address of another variable. An array that stores only addresses of other variables are known as an array of pointers. Array of pointers can also contain the address of another elements of an array. Processing rules of pointers are same as that of normal arrays. The syntax for an array of pointers is:
- Datatype *arraynamr[size];
- For example:
- Int *a[3];
- It means an array of pointers a is created in memory which can contain the addresses of three integers.

declarations

```
int a[5] = {1,2,3,4,5};
int *p = &a[0];
```

	Memory address	Memory contents
a	4000	1
	4004	2
	4008	3
	4012	4
	4016	5
p	4020	4000

Pointer and Multidimensional arrays

- It is possible to use pointers with two or more dimensions. The two dimensional array is called a matrix. A simple two-dimensional array element is defined in the form:
- A[i][j]
- Where ‘A’ is the array name
- The 1st subscript ‘i’ represents row number.
- The 2nd subscript ‘j’ represents numbers of columns.

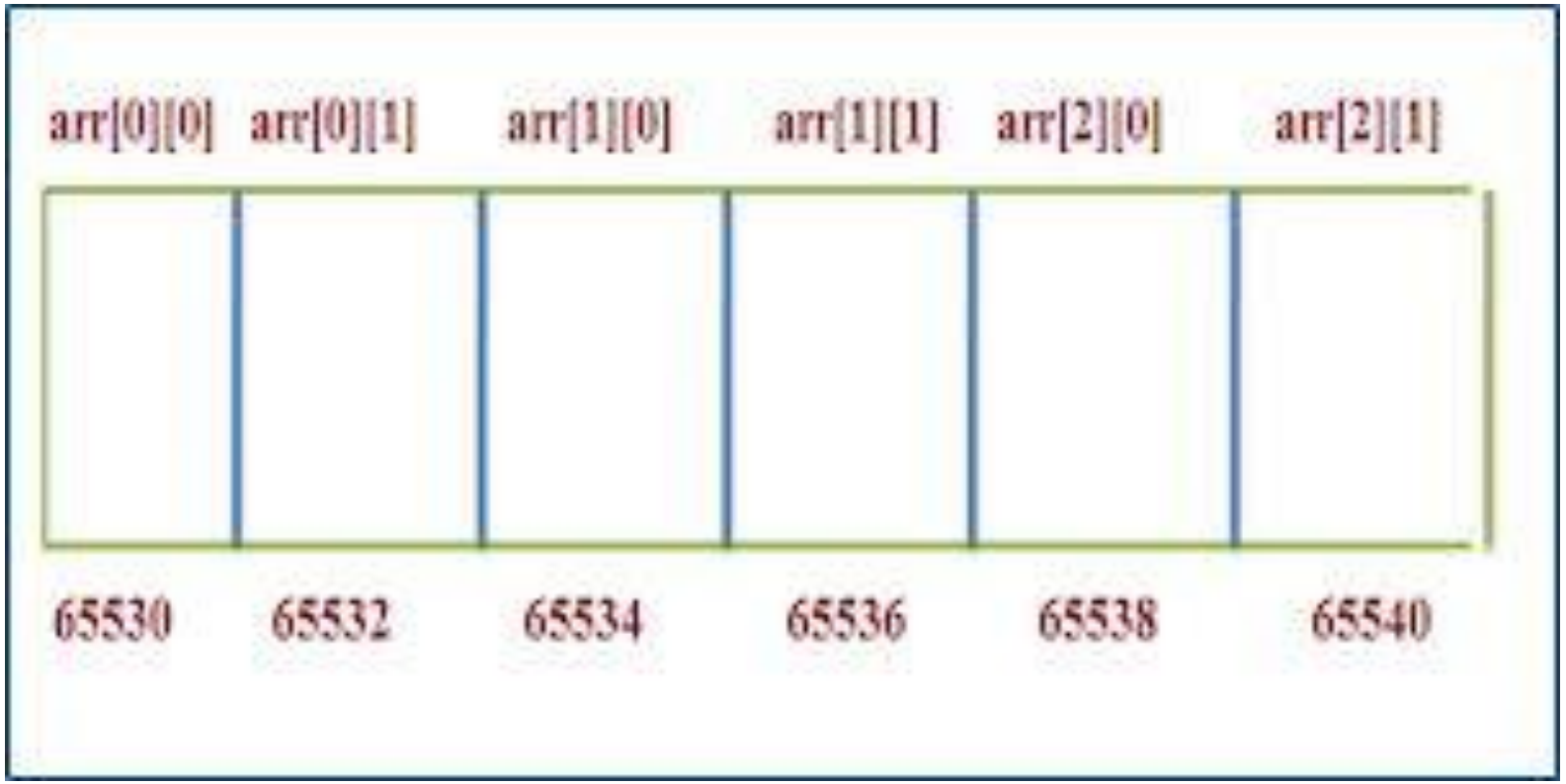
Counting of rows and columns of the two-dimensional matrix always starts with zero.
For example let initialize the elements of a two-dimensional matrix of the range 3 by 2. it will be initialized as:
Inta[3][2]= {{10 ,20}, {30, 40}, {50, 60}};
The above initialized arrays will be stored in memory in following arrangement:

Elements	**:**	**10**	**20**	**30**	**40**	**50**	**60**
Matrix location	**:**	**a[0][0]**	**a[0][1]**	**a[1][0]**	**a[1][1]**	**a[2][0]**	**a[2][1]**
Address	**:**	**200**	**202**	**204**	**206**	**208**	**210**

Pointers is used in two-dimensional arrays in the same way as in single dimension .the address of the first element of the matrix can be passed on the pointer and rest of the elements can be displayed.
For example:

```
Int number[3][4] ;
Int *ptr ;
Ptr = &number[0][0] ;
```

If we increment the pointer i.e. ,ptr++ then pointer variable will be incremented to next data in the matrix.
The increment in the pointer leads to increment in the address of elements of matrix(row wise).


arr[0][0]
arr[0][1]
arr[1][0]
arr[1][1]
arr[2][0]
arr[2][1]
65530
65532
65534
65536
65538
65540

ARRAY OF STRUCTURE

STRUCTURE

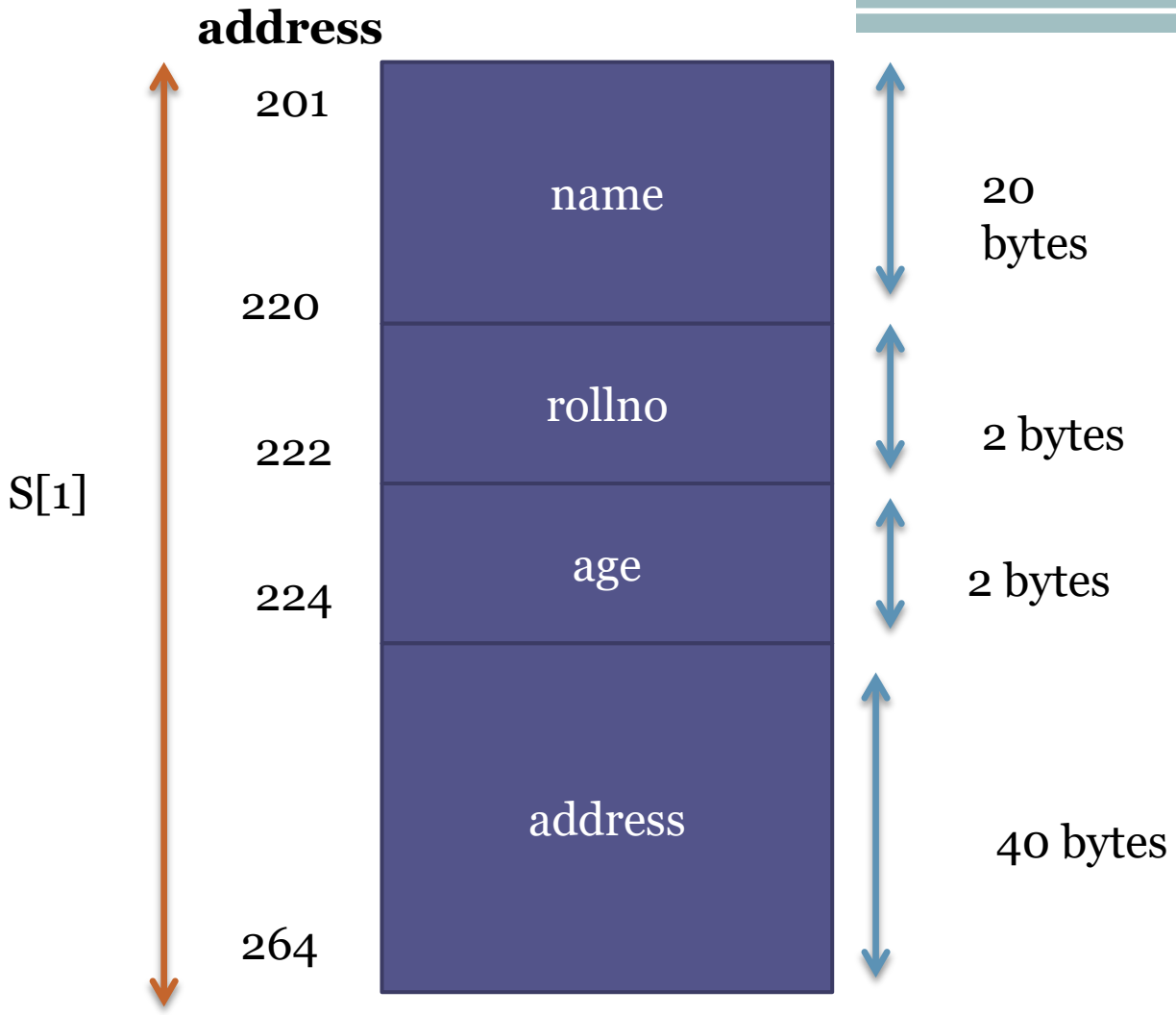
- A structure is a collection of related data items held together in a single unit and is referenced by a single name. the data items can be of different data types. The data items enclosed with in a structure are known as structure members or element or fields.

Structure declaration

- A structure groups various variable into a single record. A structure declaration specifies the grouping of variable of different types in a single unit.
- The syntax of declaring a structure in ‘C’is
- Struct structure name
- {
- Datatype variablename1;
- Datatype variablename2;
- ....
- .....
- ...
- Datatype variablename n;
- };
- A structure declaration do not allocate any memory space.e.g.
- Struct student
- {
- Char name[20];
- Int age,roll no;
- Char address[40] ;
- };

Structure definition

- Structure definition creates structure variables and allocate storage space for them. no storage space is allocated at the time of structure declaration. All the structure members are allocated contiguous space in the memory.
- Struct student s1
- Data type variable declaration
- Creates one variable s1 of the data type and allocates memory to its members as


address
201
220
222
224
264
S[1]
name
rollno
age
address
20
bytes
2 bytes
2 bytes
40 bytes

ARRAY AND STRUCTURES

- Structure and array can be used together in two way:

- 1.Structure containing array
- 2. Array of structure

Structure containing array

- structure containing array means some members of a structure is a collection of same type e.g. consider an example of a structure “student”
- Storing marks of 5 subjects of a student. It can be represented as
- struct student
- {
- Int roll no;
- Int marks[5];
- Float percent;
- }s1;
- The individual marks can be accessed as
- S1.marks[0],s1.marks[1],s1.marks[2]

ARRAY OF STRUCTURES

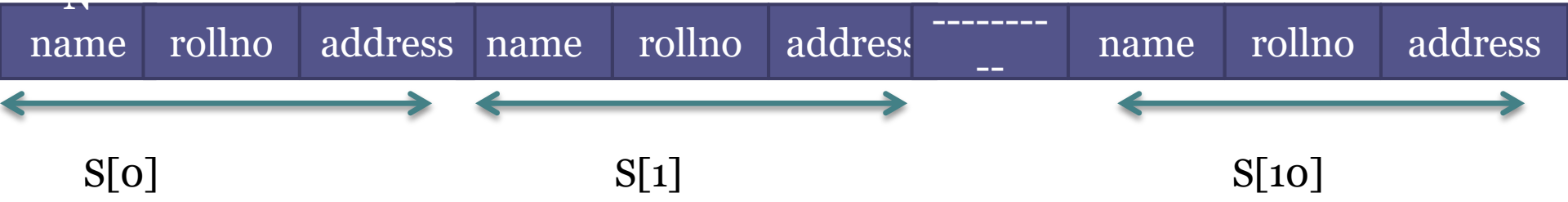
- An array of structure refers to an array in which each element is a structure. as an array of 5 integers is a collection of 5 integers values ,similarly, an array of 5 student[structure] is a collection of details of 5 student structure type. Consider an example of student structure:

- Struct student
- {
- Char name[15];
- Int rollno;
- Char adress;
- };
- Student data [50];
- The above declaration sets the memory space for 50 objects.

The defination of such an array will be

Student s[10]
Datatype array of 10 elements of type student

S[0] will refer to student1
S[1] will refer to student2 and so on.
Memory representation is contiguous as in the case of array of basic data types . E.g. student
S[10] occupies contiguous memory for all elements as follows:



Memory representation of array of structures

INITIALIZING ARRAY OF STRUCTURES

- The arrays of structure can also be initialized same as array data in C.it is to be noted that only static or external variable can be initialized. A simple example of initializing the array objects of a structure is :
- Struct employee
- {
- Char name[15];
- Int number;
- Int salary;
- };

- Employee data[3]= {{‘rohan”,1,20000}, {’ sumit’,2,20000}, {‘prem’,3,20000}};
- The above initialized values will be assigned by the compiler as:
- Data[0].name=‘rohan’ data[0].number=1 data[0].salary=20000
- Data[1].name=‘sumit’ data[1].number=2 data[1].salary=20000
- Data[2].name=‘prem’ data[2].number=3 data[2].salary=20000
- If in case the structure object or any elements of structure are not initialized the compiler will automatically assign zero to the field of that particular record .for example for above
- Employee data[3]= {{‘rohan’,1}, {’ sumit’,2}, {‘prem’,3}};
- The values assigned by the compiler for the above declaration will be
- Data[0].name=‘rohan’ data[0].number=1 data[0].salary=0
- Data[1].name=‘sumit’ data[1].number=2 data[1].salary=0
- Data[2].name=‘prem’ data[2].number=3 data[2].salary=0

Sparse matrix & dense matrix

- Sparse matrix: contains mostly zero elements.
- Such as

$$A = \begin{bmatrix} 0 & 2 & 0 & 0 & 3 \\ 2 & 0 & 0 & 0 & 0 \\ -1 & -2 & 0 & 4 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

DENSE MATRIX:containing less 0 elements.

$$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 2 & 5 \\ 2 & 1 & 0 \end{pmatrix}$$

LIMITATION OF LINEAR ARRAYS

The main limitations of linear arrays are

1. The prior knowledge of number of elements in the linear array is necessary
2. These are static structures. Static in the sense that memory is allocated at compilation time their memory used by them cannot be reduced or extended.
3. Since the elements of these arrays are stored in the these arrays are time consuming this is because of moving down or up to create a space of new element or to occupy the space vacated by the deleted element.