Lecture 4

In this lecture, we will learn

1. Pointers and Structures
2. Pointers and 2-dim arrays
3. Sparse(**<인구 등이>** 희박한**,** 성긴**,** 드문드문한**(opp. dense)**) Matrices

Structures

In one of previous(이전의) lectures,we learn that data types can be classified(분류된**;<**광고가**>** 항목별의) into 2 types.

1. Primitive (원시의**,** 초기의**;**태고의**,** 옛날의)
2. User Defined Data type

Primitive data type are int, char, float, etc provided by the language.

User defined data type is what we (user) make depending on our requirements (요구물**,** 필요물**,** 필수품
). For example, I want a data type which can store information (정보**,** 보도**,** 소식**;**자료) about an employee. Now, for this requirement, I can create my own (**[**소유의 뜻을 강조하여**]** 자기 자신의) data type. We use **structure** for creating user defined data type. Structures in C are very much similar (유의어) to Classes in C++.

We are now going to create a user defined data type which can store all the information about an employee. Every employee has the following information

1. Last Name
2. First Name
3. Salary

```
Structure employee
{
	char lastName[20];
	char firstName[20];
	float salary;
};
```

Now, employee is a **user defined data type**. We can create a variable of type employee like we create variable of type int or char or float.

lastName, firstName, salary are called as **structure members**.

```
employee e;
```

C compiler allocates (배분하다) 44 bytes (20+20+4)for 'e'. We can use the sizeOf(employee) to find the amount of memory allocation.

Write a program to read a variable of type employee and then print its value.

```
void main()
{
        Structure employee
        {
                char lastName[20];
                char firstName[20];
                float salary;
        };

        struct employee e1;

        printf("\nEnter last name of the employee");
        scanf("%s",e.lastName);

        printf("\n Enter first name of the employee");
        scanf("%s", e.firstName);

        printf("\nEnter salary of the employee");
        scanf("%f", &e.salary);

        printf("\nDetails of employee are");
        printf("\nLastName = %s", e.lastName);
        printf("\nFirstName = %s", e.firstName);
        printf("\nsalary = %f", e.salary);

}
```

One thing that is important to notice in the above program is, How we access (장소•사람 등에 의) the structure members?
We can access structure members of a structure by using a '**.**' operator. For example, to access lastName, we use e.lastName.

Next topic is, How can we use pointers with Structures. Well, it is quite easy and same as we use pointers with variables. To declare a pointer to a structure, syntax is

```
struct employee *e2;
e2 = &e1;
```

To access the structure members using pointer, syntax is

```
e2-> lastName
or
(*e2).lastName
```

Here is a program that prints the value of structure members using pointers.

```
void main()
```

```
{
        Structure employee
        {
                char lastName[20];
                char firstName[20];
                float salary;
        };

        struct employee e1;
        struct employee *e2

        printf("\nEnter last name of the employee");
        scanf("%s",e.lastName);

        printf("\n Enter first name of the employee");
        scanf("%s", e.firstName);

        printf("\nEnter salary of the employee");
        scanf("%f", &e.salary);

        e2 = &e1;
        printf("\nDetails of employee are");
        printf("\nLastName = %s", e2->lastName);
        printf("\nFirstName = %s", e2->firstName);
        printf("\nsalary = %f", e2->salary);
```

Pointers and 2-dim arrays

We know that 2-dim arrays are the one that has 2 dimensions. For example

```
char ch [rows] [cols];

or

char ch [5][10];
```

Above 2-dim array declaration can also be stated like this, there are 5 arrays and each array is of size 10. All the five arrays have a common name 'ch' and has fixed size 10.

Assume we have filled (채우다) this two dimensional array with data of some kind. In memory, it might look as if it had been formed by initializing (디스크·내부 기억 장치 등을) 5 separate (가르다, 떼다, 분리하다) arrays using something like:

```
ch[0] = {'0','1','2','3','4','5','6','7','8','9'}
ch[1] = {'a','b','c','d','e','f','g','h','i','j'}
ch[2] = {'A','B','C','D','E','F','G','H','I','J'}
```

ch[3] = {'9','8','7','6','5','4','3','2','1','0'}
ch[4] = {'J','I','H','G','F','E','D','C','B','A'}

So, there are 5 arrays, each of size 10 and all referred (알아보도록 하다, 조회하다) by a common (공통의, 공동의, 공유의) name ch.

Individual elements can be referred (알아보도록 하다, 조회하다) as
ch[0][0] = '0';
ch[1][2] = 'c';
etc

if base address is 100
*(ch + 0) or ch[0], is 100 or the address of 1st row or the address of 1st element &ch[0][0]
*(ch + 1) or ch[1], is 110 or the address of 2nd row or the address of 11th element &ch[1][0]
*(ch + 2) or ch[2], is 120 or the address of third row or the address of 21st element &ch[2][0]
*(ch+3) or ch[3], is 130 or the address of 4th row or the address of 31st element &ch[3][0]
*(ch+4) or ch[4], is 140 or the address of 5th row or the address of 41st element &ch[4][0]

i.e. *(ch+i) or ch[i] is the addres of ith row.

The following program prints the address of rows of a 2-dim array.

```
void main()
{
        char ch[5][10] = {
                                {'0','1','2','3','4','5','6','7','8','9'},
                                {'a','b','c','d','e','f','g','h','i','j'},
                                {'A','B','C','D','E','F','G','H','I','J'},
                                {'9','8','7','6','5','4','3','2','1','0'},
                                {'J','I','H','G','F','E','D','C','B','A'}
                        };

        printf("address of rows are");
        for(i=0;i<5;i++)
                printf("\n address of row %d = %u is", i, *(a+i));

}
```

So, what is

(*(ch+0)+2), it is 102 or the address of (1st row, 3rd col) or &ch[0][2]
(*(ch+3)+5), it is 135 or the address of (4th row, 6th col) or &ch[3][5]

i.e.(*(ch+i)+j), it is the address of (ith row, ith col) or *ch[i][j].

The following program prints the addresses of all the elements of

```
void main()
{
        char ch[5][10] = {
                                {'0','1','2','3','4','5','6','7','8','9'},
                                {'a','b','c','d','e','f','g','h','i','j'},
                                {'A','B','C','D','E','F','G','H','I','J'},
                                {'9','8','7','6','5','4','3','2','1','0'},
                                {'J','I','H','G','F','E','D','C','B','A'}
                        };

        printf("addresses are");
        for(i=0;i<5;i++)
                for(j=0;j<10;j++)
                        printf("\n address of a[%d][%d] = %u is", i,j, (*(a+i)+j));
}
```

And
((a+i)+j) is equivalent to a[i][j].

Program for printing all the elements of the 2-dim array

```
void main()
{
        char ch[5][10] = {
                                {'0','1','2','3','4','5','6','7','8','9'},
                                {'a','b','c','d','e','f','g','h','i','j'},
                                {'A','B','C','D','E','F','G','H','I','J'},
                                {'9','8','7','6','5','4','3','2','1','0'},
                                {'J','I','H','G','F','E','D','C','B','A'}
                        };

        printf("Value of elements are");
        for(i=0;i<5;i++)
                for(j=0;j<10;j++)
                        printf("\n  a[%d][%d] = %d is", i,j,  *(*(a+i)+j));
}
```

Matrices

In mathematics, matrix is a rectangular table of numbers. The horizontal lines in the matrix are called as rows and vertical lines are called as columns. We can represent matrix using a 2-dim arrays.

Some basic definitions related to matrices are.

1. Matrix is said to be **square** if number of rows is equal to the number of columns . i.e. **m=n.**
2. When only the diagonal elements of the square matrix are 1, and every other element is zero, it is called as **Identity matrix.**

3. In a square matrix, position wise the elements are divided as :
    - Diagonal elements
    - Lower triangle, i.e. elements below the diagonal.
    - Upper triangle, i.e. elements above the diagonal.

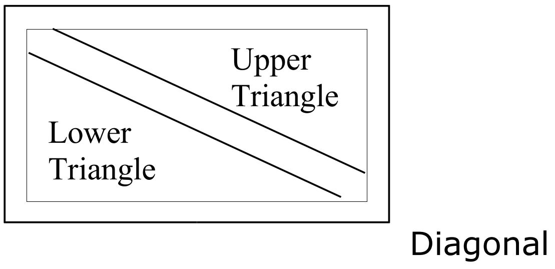

Fig 3. two dimensional matrix with classifications.

4. Symmetric ((좌우) 대칭적인, 상칭적인) matrix is a square matrix whose transpose (<위치·순서를> 바꾸어 놓다[넣다) is identical to the original matrix.
5. Transpose of the matrix is changing the row elements to the column elements. i.e. the element in the (i,j) position will occupy (j,i) position in the transpose. The transpose matrix will have the size n x m.

Sparse Matrix

A matrix in which majority (대부분, 대다수, 태반) of the elements are zeros (0) is known as sparse matrix.

In scientific calculations, a matrix with hundreds of row and column are used. Most of the elements are 0 in these matrix. For example, in a 10x10 matrix, only 15 elements are nonzero, remaining 85 elements are zeros. To save space, we can store a matrix in a different form using 1-dim arrays.

Let's see how it works.

We will make a User Defined Data-Type using structure. This structure will contain three elements

1. Row number
2. Column number
3. Value

```
Structure sparse
{
        int row;
        int col;
        int value;
}
```

Make a 1-dim array of data-type sparse.

Structure sparse s1[size]; // size is declared as required.

We take an example to understand the concept of sparse matrices.

$$\text{matrix A} \quad \begin{array}{c|ccccc} \text{Row/Col} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 2 & 0 & 0 & -3 & 0 \\ 1 & 0 & 0 & 11 & 0 & 0 \\ 2 & 0 & -7 & 0 & 0 & 1 \\ 3 & -4 & 0 & 0 & 0 & 9 \end{array}$$

fig 4. matrix of order 4 X 5

Above is a 4x5 sparse matrix which has only 7 values, remaining 13 values are 0. We can represent the above matrix in a different data structure in order to save some space.

Structure sparse s1[8];

s1[0].row = 4, s1[0].col=5, s1[0].value=7

First entry of the array contains number of rows (4), number of columns (5) and number of nonzero elements (7).

Second and subsequent (다음의, 그 후의;버금가는) entries contain the row, col, value.

s1[1].row=0, s1[1].col=0, s1[1].value=2

s1[2].row=0, s1[2].col=3, s1[2].value=-3

s1[3].row=1, s1[3].col=2, s1[3].value=11

s1[4].row=2, s1[4].col=1, s1[4.value=-7

s1[5].row=2, s1[5].col=4, s1[5].value=1

s1[6].row=3, s1[6].col=0, s1[6].value=-4

Sparse Representation:

	s1[0]	s1[1]	s1[2]	s1[3]	s1[4]	s1[5]	s1[6]	s1[7]
row	4	0	0	1	2	2	3	3
col	5	0	3	2	1	4	0	4
val	7	2	-3	11	-7	1	-4	9

sparse representation

Here the position 0 in the sparse matrix representation actually tell about the total number of rows , total number of columns and number of non-zero elements.

Below, is a program to represent a sparse matrix using 1-dim array.

```
void main()
{
        int a[4][5] = {
                                {2, 0, 0, -3, 0},
                                {0, 0, 11, 0, 0},
                                {0, -7, 0, 0, 1},
                                {-4, 0, 0, 0, 9}
                        };

        Structure sparse
        {
                int row;
                int col;
                int value;
        }
        Structure sparse s[8];
        int i,j;
int k=1;
for(i=0;i<4;i++)
{
        for(j=0;j<5;j++)
        {
                if(a[i][j] != 0)
                {
                        s[k].row = i;
                        s[k].col = j;
                        s[k].value= a[i][j];
                        k++;
                }
        }
}
        s[0].row = 4;
        s[0].col = 5;
        s[0].value = k;
}
```

Addition of two sparse matrices

First we need to decide the size of resultant (결과, 물리】 합력, 합성 운동) sparse matrix, i.e how many rows and columns are there. For example, if there are 2 sparse matrices , one sparse matrix 'x1' of size 3x4, and second sparse matrix 'x2'of size 4x3.

```
int x1[3][4] = {
                        {0, 0, 1, 0},
                        {2, 0, 0, 0},
                        {0, 4, 3, 0},
                };
```

Corresponding representation of sparse matrix 'x1' is

s1[0].row=3, s1[0].col=4, s1[0].value=4
s1[1].row=0, s1[1].col=2, s1[1].value=1

s1[2].row=1, s1[2].col=0, s1[2].value=2
s1[3].row=2, s1[3].col=1, s1[3].value=4

s1[4].row=2, s1[4].col=2, s1[4].value=3

2nd sparse matrix

```
int x2[4][3] = {
                    {5, 0, 0},
                    {2, 0, 3},
                    {0, 0, 1},
                    {0, 4, 0}
             };
```

Corresponding representation of sparse matrix 'x2' is

s2[0].row=4, s2[0].col=3, s2[0].value=5
s2[1].row=0, s2[1].col=0, s2[1].value=5
s2[2].row=1, s2[2].col=0, s2[2].value=2
s2[3].row=1, s2[3].col=2, s2[3].value=3

s2[4].row=2, s2[4].col=2, s2[4].value=1

s2[5].row=3, s2[5].col=1, s2[5].value=4

We will create another structure variable 's3'. Number of rows in s3 will be greater of the s1[0].row and s2[0].row

i.e. s3[0].row = max(s1[0].row,s2[0].row)

s3[0].row = max(3,4)

s3[0].row = 4

Similarly for columns in s3

s3[0].col = max(s1[0].col, s2[0].col)

s3[0].col = max(4,3)

s3[0].col = 4

So,

s3[0].row = 4, s3[0].col = 4, s3[0].value = ?

Step 2: Next, we will check whether([간접의문문의 명사절을 이끌어] ...인지 어떤지) rows numbers of s1 and s2 are same. If not same, then copy value from lower row number.

If row numbers of s1 and s2 are same, then column numbers of s1 and s2 are compared. If column numbers are also same, then add the values

i.e. S3[k].value = s1[i].value + s2[i].value

If row numbers of s1 and s2 are same but column numbers of s1 and s2 are not same, then copy value from lower column number.

For ex:

we will find

```
while(i<s1[0].row && j<s2[0].row)
{
    if (s1[i].row == s2[j].row) // if row is same
        {
            if (s1[i].col == s2[j].col) // if row is same and col is same
            {
                s3[k].row = s1[i].row;
                s3[k].col = s1[i].col;
                s3[k].value = s1[i].value + s[j].value;
                i++;
                j++;
                k++
            }
            else if (s1[i].col < s2[j].col) // if row is same, but s1[i].col < s2[j].col
            {
                s3[k].row = s1[i].row;
                s3[k].col = s1[i].col
                s3[k].value = s1[i].value;
                k++;
                i++;
            }
            else if (s2[j].col < s1[i].col) // if row is same, but s2[j].col < s1[i].col
            {
                s3[k].row = s2[j].row;
```

```
                        s3[k].col = s2[j].col
                        s3[k].value = s2[j].value;
                        k++;
                        j++;
                }
        }
        else if (s1[i].row < s2[j].row) // rows are not same, and s1[i].row < s2[j].row
        {
                s3[k].row = s1[i].row;
                s3[k].col = s1[i].col
                s3[k].value = s1[i].value;
                k++;
                i++;
        }

        else if (s2[j].row < s1[i].row)
        {
                s3[k].row = s2[j].row;
                s3[k].col = s2[j].col
                s3[k].value = s2[j].value;
                k++;
                j++
        }
}

// if first matrix ends before second matrix end
while (j < s2[0].row)
{
                s3[k].row = s2[j].row;
                s3[k].col = s2[j].col
                s3[k].value = s2[j].value;
                k++;
                j++
}
```

```
// if second matrix ends, before first matrix end
while (i < s1[0].row)
{
                    s3[k].row = s1[i].row;
                    s3[k].col = s1[i].col
                    s3[k].value = s1[i].value;
                    k++;
                    i++;
}

s3[0].value = k;
```

Program for addition of 2 sparse Matrices

Structure s3 will be

```
s3[0].row=4, s3[0].col=4, s3[0].value=7
s3[1].row=0, s3[1].col=0, s3[1].value=5
s3[2].row=0, s3[2].col=2, s3[2].value=1
s3[3].row=1, s3[3].col=0, s3[3].value=4
s3[4].row=1, s3[4].col=2, s3[4].value=3
s3[5].row=2, s3[5].col=1, s3[5].value=4
s3[6].row=2, s3[6].col=2, s3[6].value=4
s3[7].row=3, s3[7].col=1, s3[7].value=4
```