Multiple Dimension Arrays

Multiple Dimension Arrays

- A *multiple dimension array* is an array that has two or more dimensions.
- Two dimensional arrays are the simplest type of multiple dimension arrays. They can be used to represent tables of data, matrices, and other two dimensional objects.
- One of the most obvious example of a two dimensional array is a table.

Multi-dimensional ARRAYS, Cont.

- When might we use a multi-dimensional array?

- Array of names (i.e., array of strings)
- This generalizes to ***a list of lists***
- Other examples,

  Checkerboard

  Matrices, lists of vectors

Multi-Dimensional Arrays

```
#define X_AXIS 10
#define Y_AXIS 10
#define Z_AXIS 10

int array1d[X_AXIS];
int array2d[X_AXIS][Y_AXIS];
int array3d[X_AXIS][Y_AXIS][Z_AXIS];

array2d[5][6] = 4;
```

Show some C code to initialise array2d to all zeros
(Hint: need two (nested) for loops)

Two Dimensional Arrays

- We can see how a two dimensional array looks like a table or matrix.
- Thus, we can represent a two dimensional array as rows and columns.
- To declare a two-dimensional array:

(data type) array_name[# of rows] [# of columns];

2-Dimensional Arrays

Can be illustrated as a table:

Each row is a separate array.

They could even have different lengths.

	col 0	**col 1**	**col 2**	**col 3**
row 0	**1**	**2**	**3**	**4**
row 1	**5**	**6**	**7**	**8**
row 2	**9**	**10**	**11**	**12**

Two Dimensional Arrays Declarations

- Therefore, a two dimensional array of integers, named x, with 3 rows and 4 columns would be declared as:

```
int x[3][4];
```

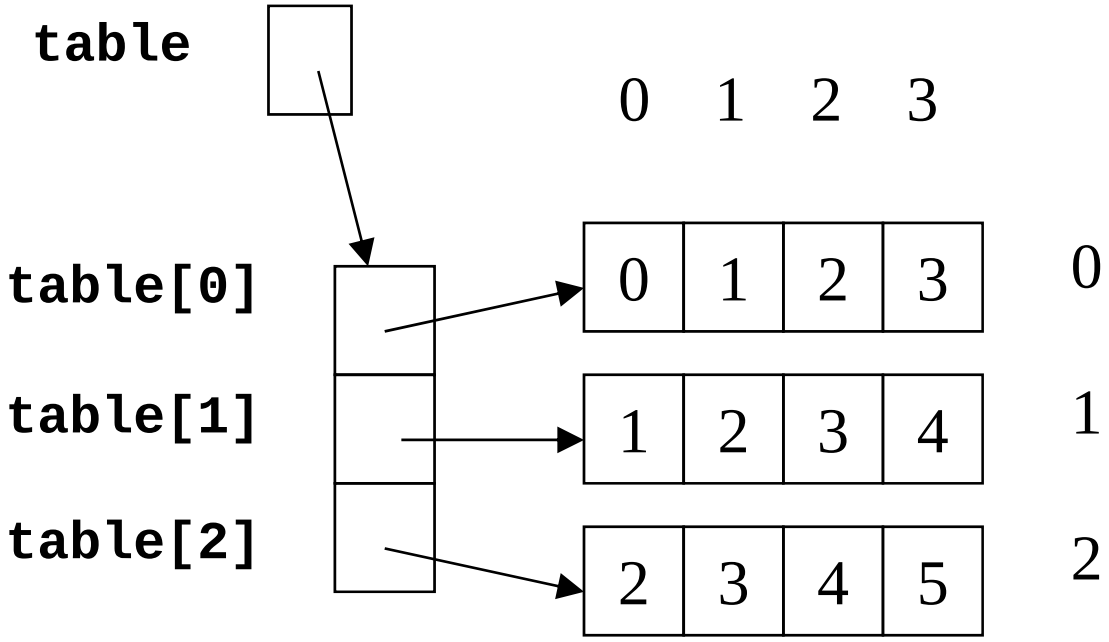
- A two dimensional array of characters, named c, with 119 rows and 2 columns would be declared as:

```
char c[119][2];
```

Two-Dimensional Arrays

```
# define MAXSTU 3
# define MAXSUB 4
int table[MAXSTU][MAXSUB]

for (row = 0; row < MAXSTU; row++)
   for (col = 0; col < MAXSUB; col++)
     scanf ("%d",&table[row][col])
```



Multiple Dimension Array Declarations

- We can also declare arrays of more than two dimensions. A six dimensional array of doubles could be declared as:

```
double a[2][3][9][4][5][3];
```

- A three dimensional array of integers could be declared as:

```
int x[4][5][4];
```

Initialization of Multiple Dimension Arrays

- Multiple dimension arrays can be initialized in much the same way as one dimensional arrays.

```
char board[3][3] = { {‘X’, ‘X’, ‘O’},
                     {‘O’, ‘X’, ‘O’},
                     {‘X’, ‘O’, ‘X’} };
```

- Notice how the initialization list is grouped into rows.

Initialization of Multiple Dimension Arrays

- In addition to grouping the initialization list into rows, the initialization list can simply be provided as a straight list:

```
char board[3][3] = {‘X’,‘X’,‘O’,‘O’,‘X’, ‘O’,‘X’,‘O’,‘X’};
```

- This type of initialization list fills in the first row, left-to-right, then the second row, left-to-right, then the third row, etc…

Initialization of Multiple Dimension Arrays

char board[3][3] = {‘X’,‘X’,‘O’,‘O’,‘X’, ‘O’,‘X’,‘O’,‘X’};

- Thus, this declaration will set
  - board[0][0] to ‘X’,
  - board[0][1] to ‘X’,
  - board[0][2] to ‘O’,
  - board[1][0] to ‘O’,
  - board[1][1] to ‘X’,
  - board[1][2] to ‘O’,
  - board[2][0] to ‘X’,
  - board[2][1] to ‘O’,
  - board[2][2] to ‘X’.

Referencing Multiple Dimension Arrays

- To access an element of a multiple dimension array, we use the same form as for one dimensional arrays, with the extra dimension(s) also present, such as:

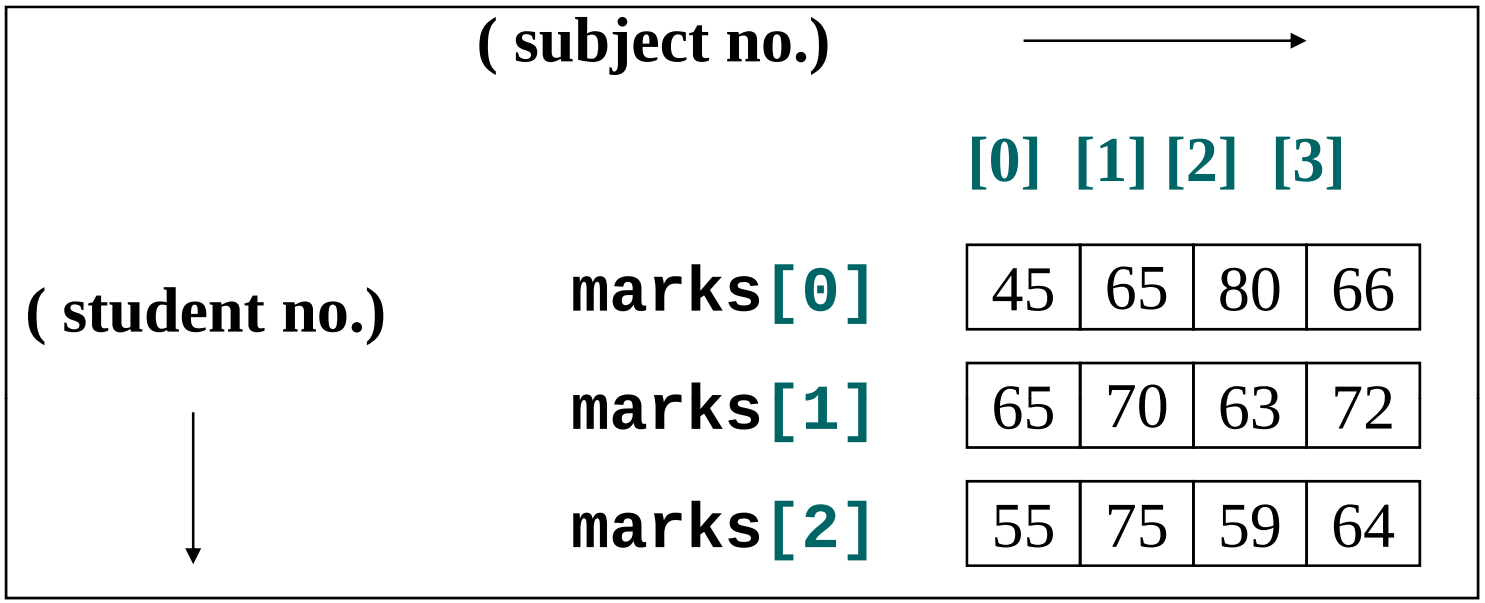
```
x[2][7] = x[1][7] + 27;
y[9][2][0][4] = 9;
printf("%d",p[3][4][5][9][0]);
```

Multi-dimensional ARRAYS

- int daytab[2][13] = {
  {0,31,28,31,30,31,30,31,31,30,31,30,31},
  {0,31,29,31,30,31,30,31,31,30,31,30,31} };
- Note initialization of arrays within an array.
- How to access an individual element
  - daytab[1][4] /* 2nd row 5th element ie30*/
  - NOTE – the following is wrong!

    daytab[1,4] /* wrong! */
- For 2-dimensional arrays, the first dimension is row, the 2nd is column. Does it matter?

Storing the marks of all students in all subjects in a two- dimensional array

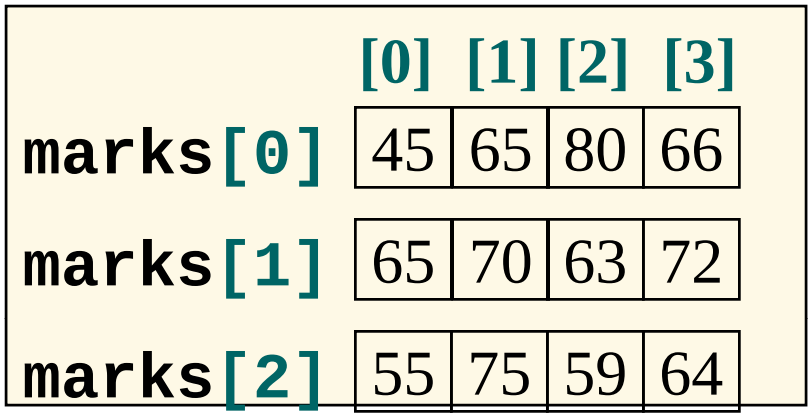


Here the marks are stored in an array
marks[students][subjects]

i.e. each row contains the marks of all subjects of one student,each column contains the marks of all the students for one subject.

Storing the marks of all students in all subjects in a two- dimensional array

/* Program to get the inputs of marks of all the students in all their subjects */

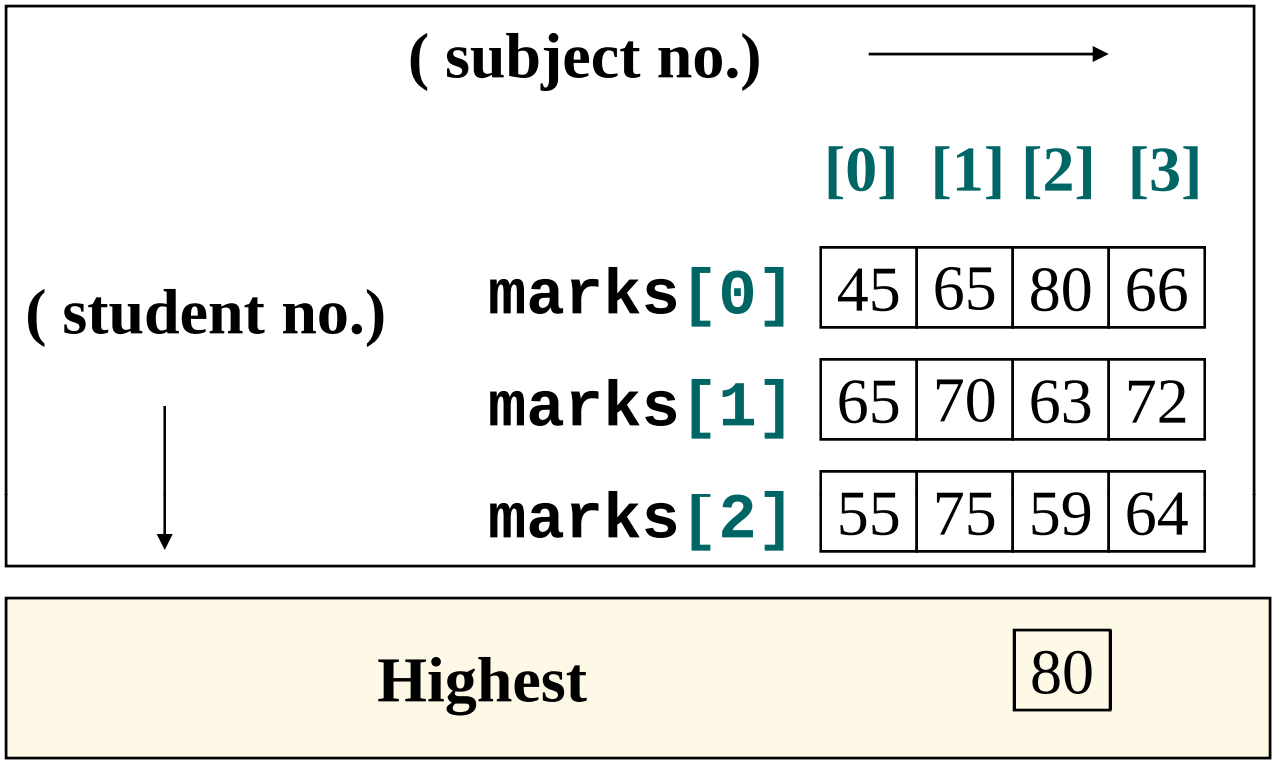


	[0]	[1]	[2]	[3]
marks[0]	45	65	80	66
marks[1]	65	70	63	72
marks[2]	55	75	59	64

```
# define MAXSTU 3
# define MAXSUB 4
main( )
{  int marks[MAXSTU][MAXSUB], sum[MAXSTU];
  int students,subjects;

/* taking the input of all the marks of all students */
  for (students = 0; students < MAXSTU; students++)
   for (subjects = 0; subjects < MAXSUB; subjects++)
     scanf (“%d”, &marks[students][subjects]);  }
```

Highest mark by any student in any subject



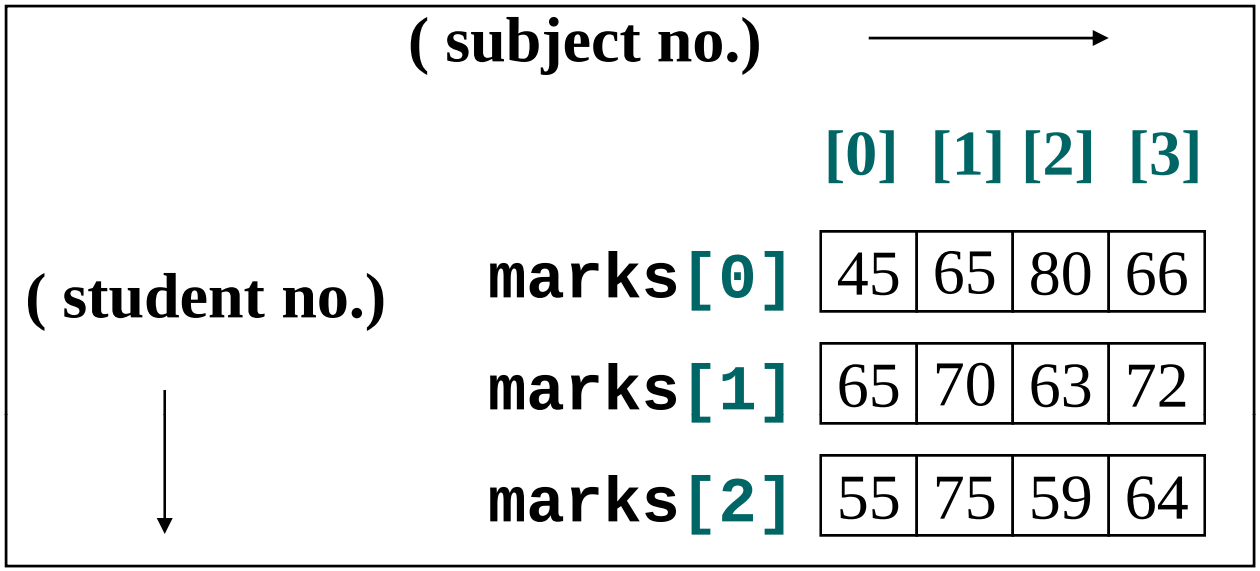
Here the marks are stored in an array
marks[students][subjects]

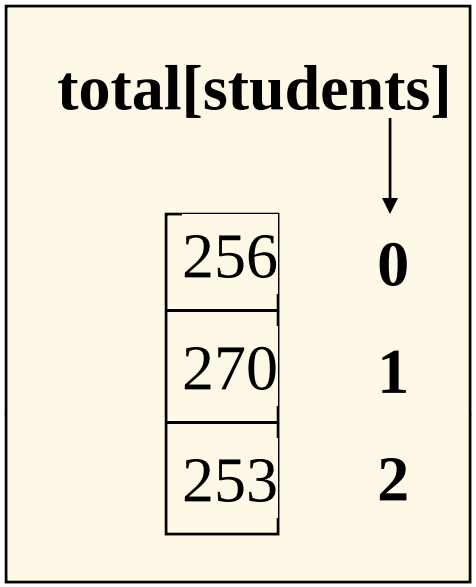
i.e. each row contains the marks of all subjects of one student.And each column contains the marks of all the students for one subject.

Highest mark by any student in any subject

```
# define MAXSTU 3
# define MAXSUB 4
main( )
{
  int marks[MAXSTU][MAXSUB], highest;
  int students,subjects;
/* to find the maximum of all the marks scored */
  highest = 0;
  for (students = 0; students < MAXSTU; students++)
  {
    for (subjects = 0; subjects < MAXSUB; subjects++ )
          {
        if (marks[students][subjects]> highest)
                highest = marks[students][subjects];
          } }
  printf ("the highest of all the subject is %d", highest);
}
```

Total marks scored by each individual student in all the subjects





Array decl. is: marks [MAXSTU][MAXSUB]
total[MAXSTU]

To store the total marks of each student, you require an array of size equal to the number of the students, total [MAXSTU]. The array size is MAXSTU because the number of students will be equal to the number of students used for the array marks.

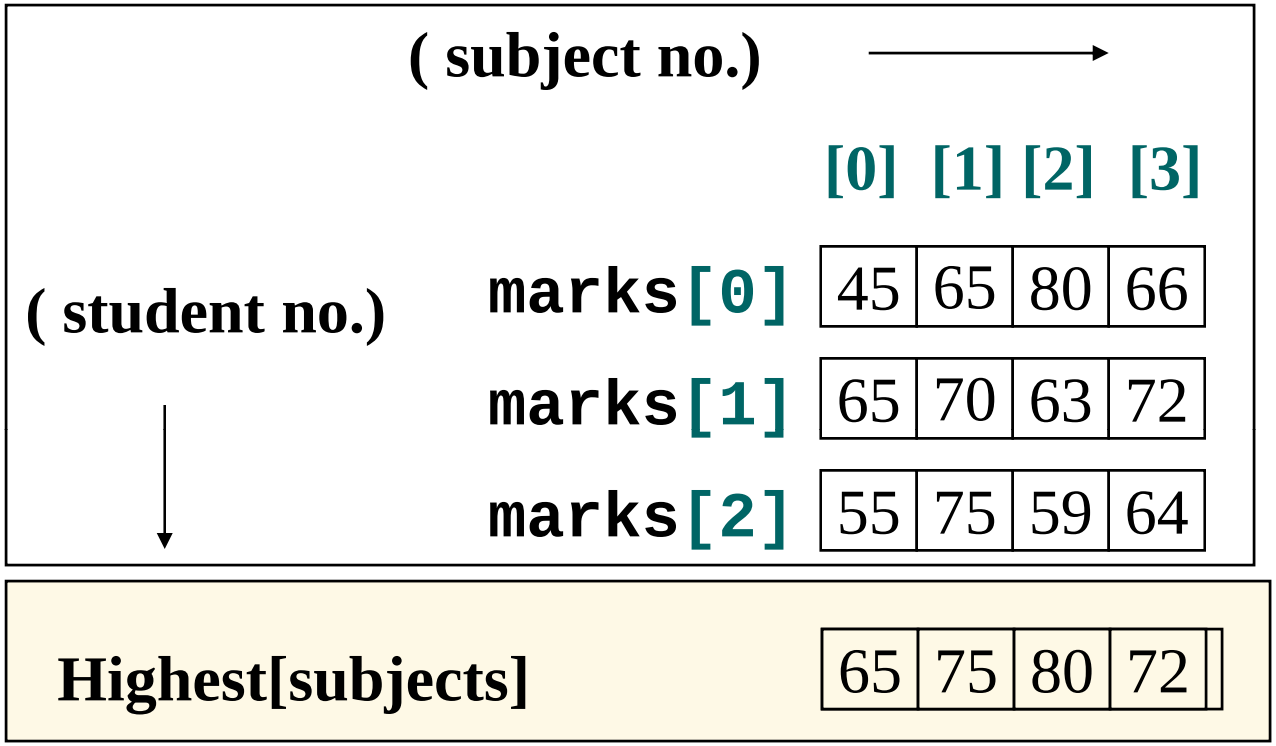
To traverse from one student to the other use the row index variable students, which is the same as the index variable used for marks array.

Total marks scored by each individual student in all the subjects

```
# define MAXSTU 3
# define MAXSUB 4
main( )
{ int marks[MAXSTU][MAXSUB], total[MAXSTU];
  int students,subjects;

/* to find the sum of all the marks scored by each
individual in all the subjects */
for (students = 0; students < MAXSTU; students++)
{  total[students] = 0;
   for (subjects = 0; subjects < MAXSUB; subjects++)
         total[students]= total[students]+
                              marks[students][subjects]);
}}
```

Highest marks scored in each individual subject



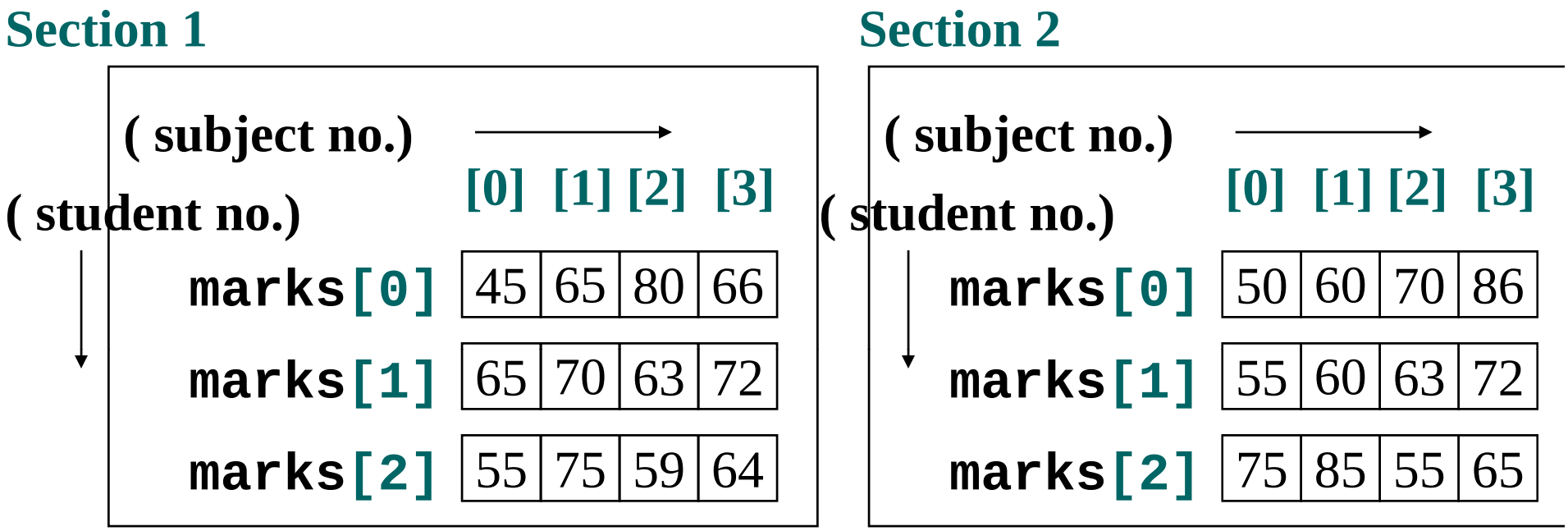
You need an array to store the highest marks, because you are storing highest marks of not just one subject but the highest marks of each individual subject. Therefore the array size would be equal to the number of subjects. Highest[MAXSUB].

To traverse from one subject to the other use the column index variable subjects.

Highest marks scored in each individual subject

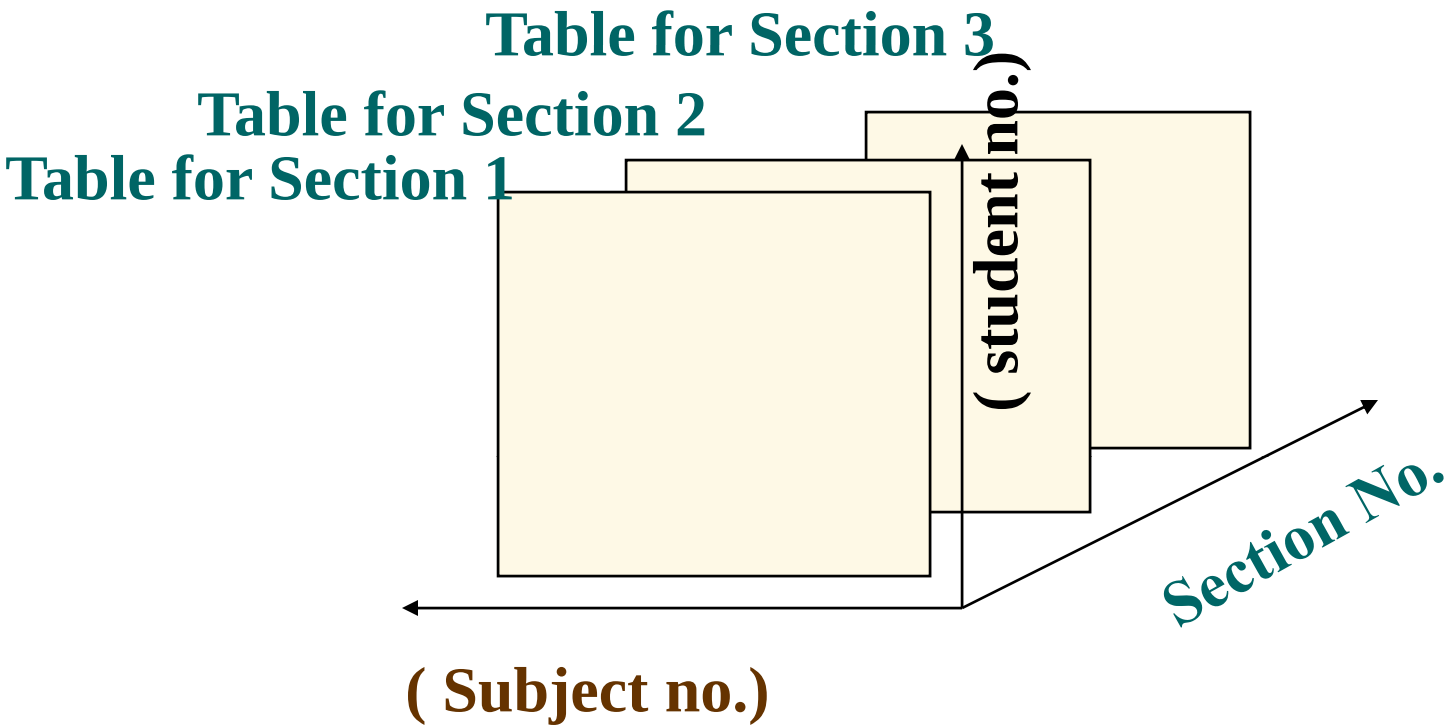
```
# define MAXSTU 120
# define MAXSUB 6
main( )
{ int marks[MAXSTU][MAXSUB], highest[MAXSUB];
  int students,subjects;
/* to find the highest marks scored in each subject */
for (subjects = 0; subjects < MAXSUB; subjects++ )
 { highest[subjects] = 0;
   for (students = 0; students < MAXSTU; students++)
         { if (marks[students][subjects]> highest[subjects])
               highest[subjects] = marks[students][subjects];
   printf ("the highest of the subject number %d is %d",
                        (subjects+1),highest[subjects]);
 }}
```

Three dimensional arrays



In the previous examples we had stored marks of all the students in all their subjects. This is only for one section. If you want to store marks for another section, one more table has to be created. Similarly for a third section, another table has to be created.

Three - Dimensional array



Three dimensional arrays can be visualized as a group of tables. Here it can be noticed that the third component is the section no. Therefore the dimensions would be

Section, Student, Subject

Three - Dimensional array

The three dimensional array would be declared as follows

marks[MAXSEC][MAXSTU][MAXSUB]

or **marks[10][30][6],** for a maximum of

10 sections. (B1-B10)

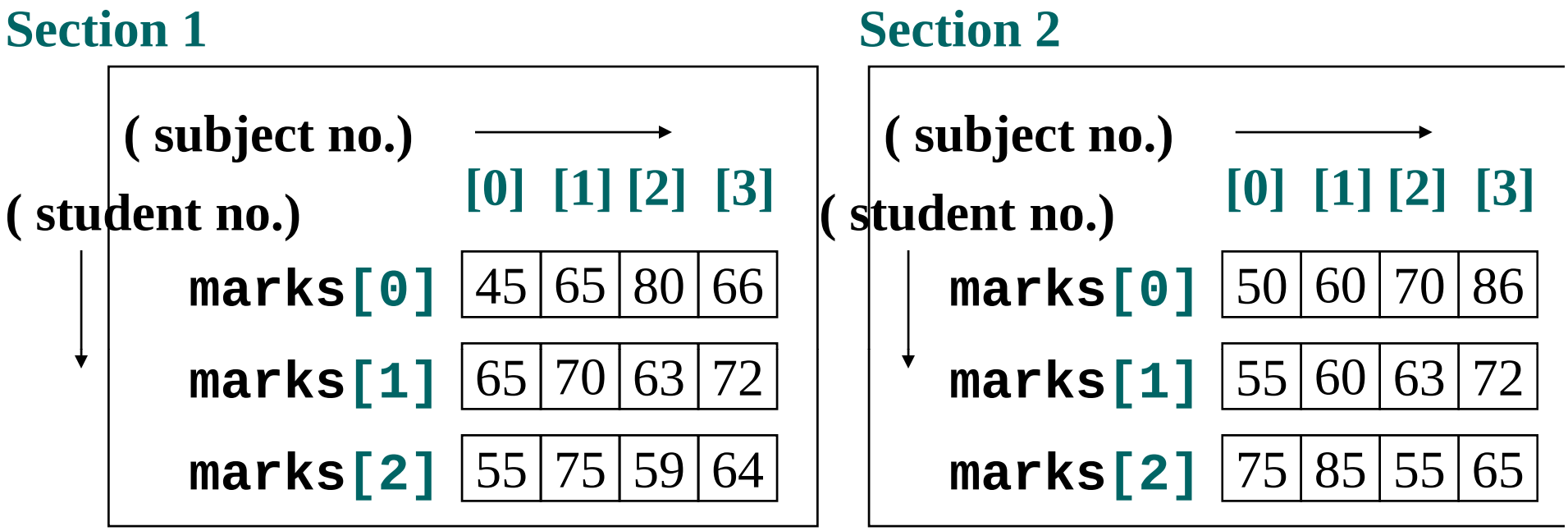
30 students.

6 subjects.

Storing the marks of all students in all the sections in all subjects in a three- dimensional array

```
# define MAXSEC 10
# define MAXSTU 30
# define MAXSUB 6
main( )
{
  int marks[MAXSEC][MAXSTU][MAXSUB];
  int sections,students,subjects;
/* taking the input of all the marks of all students */
for (sections=0; sections < MAXSEC; sections++)
  for (students = 0; students < MAXSTU; students++)
    for (subjects = 0; subjects < MAXSUB; subjects++)
      scanf (“%d”, &marks[sections][students][subjects]);
 }
```

Highest marks in any subject in any section



Highest 86

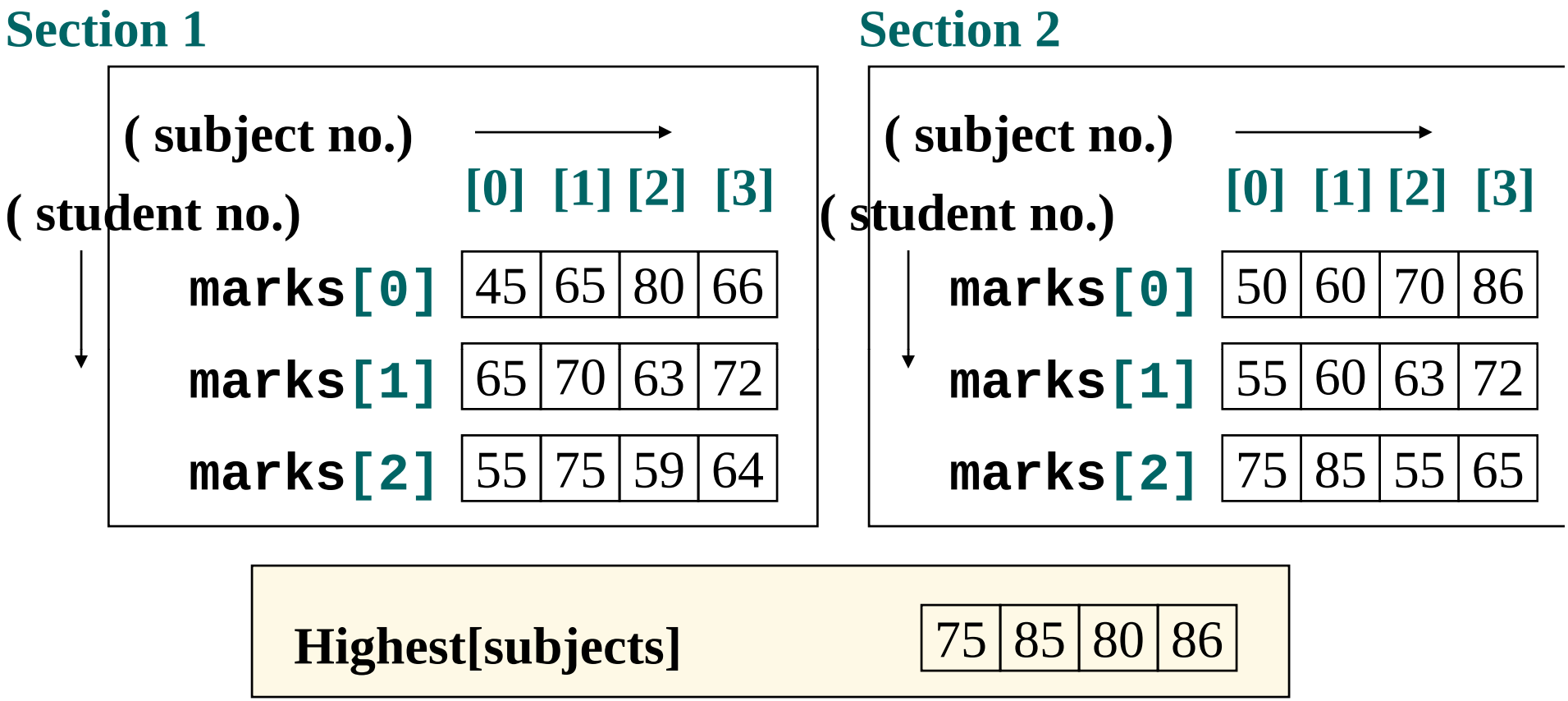
Highest marks in any subject in any section

```
/* Program to get the inputs of marks of all the students
in all their subjects for all sections*/

int highest=0;
for (sections=0; sections < MAXSEC; sections++)
  for (students = 0; students < MAXSTU; students++)
   for (subjects = 0; subjects < MAXSUB; subjects++)
    {
      if (highest < marks[sections][students][subjects])

            highest = marks[sections][students][subjects];
    }
```

Highest marks in each subject in any section



Highest marks in each subject in any section

```
int highest[MAXSUB];
for (subjects = 0; subjects < MAXSUB; subjects++)
{ highest[subjects]=0;
 for (sections=0; sections < MAXSEC; sections++)
  for (students = 0; students < MAXSTU; students++)
    {
      if (highest[subjects] <
                        marks[sections][students][subjects])

            highest[subjects] =
                        marks[sections][students][subjects];
    }
}
```

Highest marks in each subject in each section

Section 1

(subject no.) →

(student no.) ↓

	[0]	[1]	[2]	[3]
marks[0]	45	65	80	66
marks[1]	65	70	63	72
marks[2]	55	75	59	64

Section 2

(subject no.) →

(student no.) ↓

	[0]	[1]	[2]	[3]
marks[0]	50	60	70	86
marks[1]	55	60	63	72
marks[2]	75	85	55	65

Highest[sections][subjects]

65	75	80	72
75	85	70	86

Highest marks in each subject for each section

```
int highest[MAXSEC][MAXSUB];
for (subjects = 0; subjects < MAXSUB; subjects++)
 for (sections=0; sections < MAXSEC; sections++)
  { highest[sections][subjects]=0;
   for (students = 0; students < MAXSTU; students++)
    {
      if (highest[sections][subjects] <
              marks[sections][students][subjects])
             highest[sections][subjects] =
              marks[sections][students][subjects];
    }
  }
```

Highest marks of each student in each section

Section 1

(subject no.) →

(student no.) ↓

	[0]	[1]	[2]	[3]
marks[0]	45	65	80	66
marks[1]	65	70	63	72
marks[2]	55	75	59	64

Section 2

(subject no.) →

(student no.) ↓

	[0]	[1]	[2]	[3]
marks[0]	50	60	70	86
marks[1]	55	60	63	72
marks[2]	75	85	55	65

Highest[students][sections]

80	86
72	72
75	85

Highest marks of each student in each section

```
int highest[MAXSEC][MAXSTU];

for (sections=0; sections < MAXSEC; sections++)
  for (students = 0; students < MAXSTU; students++)
        highest[sections][students] = 0;
     for (subjects = 0; subjects < MAXSUB; subjects++)
        {
          if (highest[sections][students] <
             marks[sections][students][subjects])

            highest[sections][students] =
             marks[sections][students][subjects];
        }
  }
```

DISADVANTAGE OF ARRAYS

- Memory allocation of array is static:

Maximum size (maximum number of elements) requested by the programmer would be reserved in the memory irrespective of the usage of the number of elements by the user.The memory space that is unused is wasted. LESS RESOURCE UTILIZATION. For example, int test[30]

- Different data types could not be stored in an array