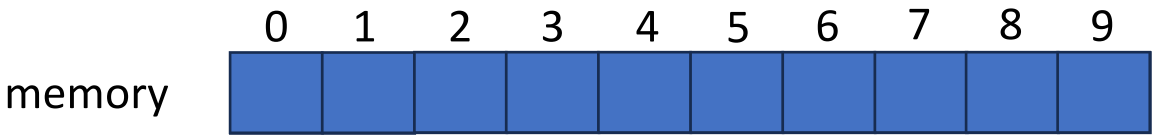
Array and memory allocations

Ch 1.2 & Ch 2

Storing grades of all students

- Number of students in class A: 10
- How?
  - Array is a map into contiguous memory locations.
  - Let’s create a 1-dimensional array with length 10.



```
int grade[10];
grade[0]=85;
…
grade[9]=99;
```

If class B has 12 students and class C has 14 students, is the program still suitable?

Dynamic memory allocation

- When writing a program, do you how much space you will need?

- When need a new area of memory, call a function “malloc” and request the amount you need.

```
int *pi;
pi = (int *) malloc(sizeof(int));
*pi = 1024;
printf(“an integer = %d\n”, *pi);
free(pi);
```

The type can also be replaced with float or char.

- The “free” function deallocates an area of memory allocated by malloc.

Storing grades of all students

- Input:
  - *N*: number of students in a class
- Dynamically allocated a 1D array to store grades.

```
int *grade;
grade = (int *) malloc(N*sizeof(int));
…
free(grade);
```

If there are 3 examinations and 5 homework assignments, is the 1-D array suitable?

2D arrays

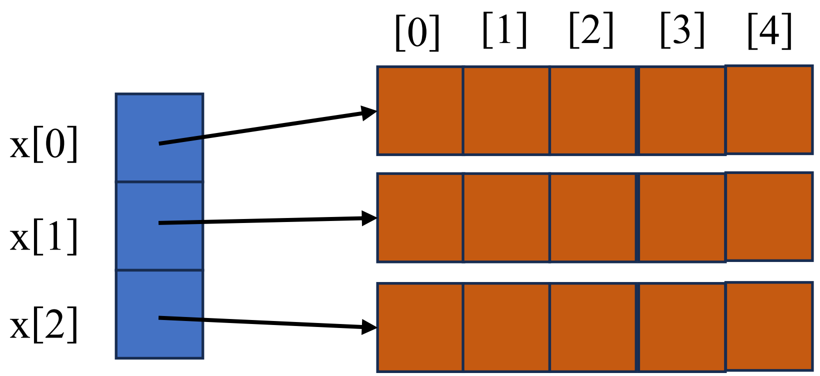
- C uses *array-of-array representation* to represent multidimensional array.

```
int x[3][5];
```

The way to access the 2D array

x[0][0]	x[0][1]	x[0][2]	x[0][3]	x[0][4]
x[1][0]	x[1][1]	x[1][2]	x[1][3]	x[1][4]
x[2][0]	x[2][1]	x[2][2]	x[2][3]	x[2][4]

C stores the 2D array in the memory in this way.



It requires four memory blocks.

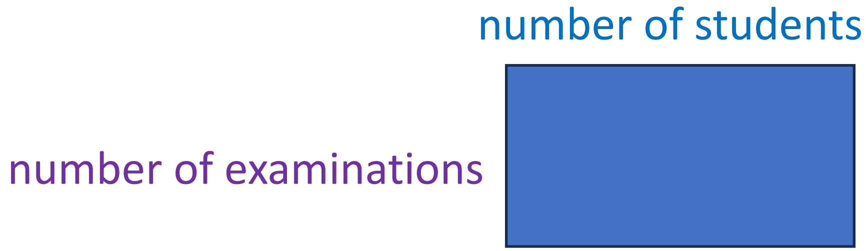
- One for the three pointers.
- Each of remaining blocks is for 5 ints.

Dynamically create a 2D array

- Input:
  - *N*: number of students in a class
  - *M*: number of examinations of each student

```
/* Declare a 2D array variable*/
int **grade;
/* Get memory for row pointers */
MALLOC(grade, M * sizeof (*grade));
/* Get memory for each row */
for (i = 0; i < M; i++)
    MALLOC(grade[i], N* sizeof(**grade));
```

```
/* Dellocate memory */
for (i = 0; i < M; i++)
    free(grade[i]);
free(grade);
```



A single column stores all scores of one student.

A single row stores all scores of one exam.

To assign scores to exam [1] for student [2]

```
grade[1][2]=80;
```

To add 2 scores to exam [0] for student [1]

```
grade[0][1]+=2;
```

Representation of 2D arrays

1. Array-of-arrays representation (as shown previously)
2. Mapping all elements of a 2D array into an ordered or linear list.

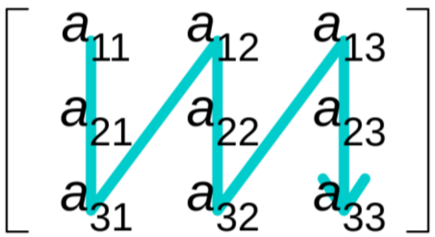
Row-major and column-major mapping

- Column-major order (default in Matlab and Fortran)
  - Elements of the columns are contiguous in memory

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Columns: From left to right

Elements: From top to bottom

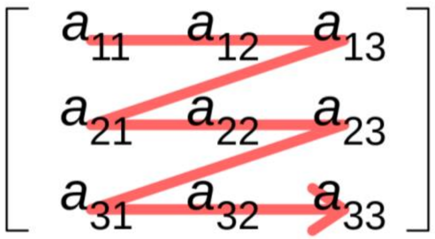

a_{11} a_{12} a_{13}
a_{21} a_{22} a_{23}
a_{31} a_{32} a_{33}

1	4	7	2	5	8	3	6	9

- Row-major order (default in C and C++)
  - Elements of the rows are contiguous in memory.

Elements: From left to right

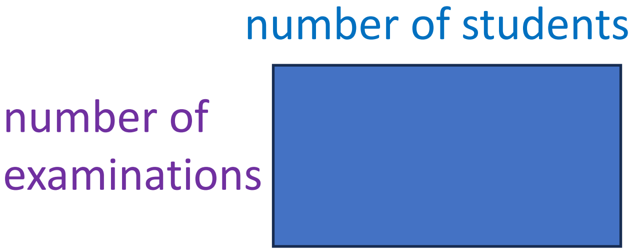
Rows: From top to bottom


a_{11} a_{12} a_{13}
a_{21} a_{22} a_{23}
a_{31} a_{32} a_{33}

1	2	3	4	5	6	7	8	9

Dynamically create a 2D array using single pointer

- Input:
  - *N*: number of students in a class (Number of columns)
  - *M*: number of examinations of each student (Number of rows)



Please reply your answers of Q1 and Q2 via the following link:

Group members: 1~3 people

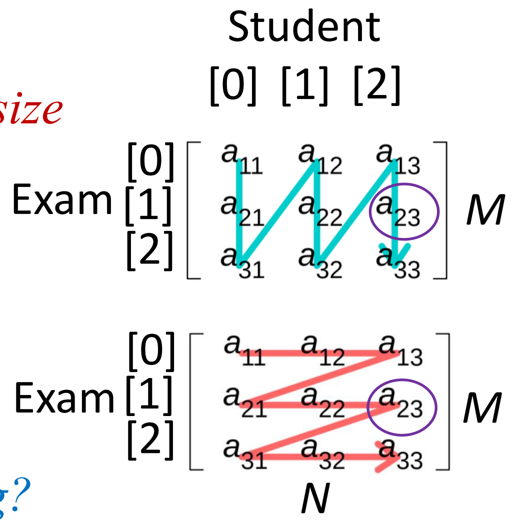
```
/* Declare a variable*/
int *grade;
/* Allocate memory of size M x N */
MALLOC(grade, M * N * sizeof(int));
```

*The address of A[i][j] is base_address+ (nRows*j + i)*size*

Assign score 80 to exam [1] for student [2]

Column major `grade[ M*2+1 ]=80;`

Row major `grade[ Q1 ]=80;`


Student
[0] [1] [2]
Exam
[0]
[1]
[2]
a_{11} a_{12} a_{13}
a_{21} a_{22} a_{23}
a_{31} a_{32} a_{33}
M
Exam
[0]
[1]
[2]
a_{11} a_{12} a_{13}
a_{21} a_{22} a_{23}
a_{31} a_{32} a_{33}
M
N

Q2: What is the addressof A[i][j] in row-major mapping?

Summary

- The way to store and create 1D arrays
    - Dynamic allocation

- The way to store and create 2D arrays
    - Dynamic allocation using multiple pointers
    - Dynamic allocation using single pointer
    - Row-major and column-major mapping