

Pointers & STL

Lab 11

Announcement

- There are no classes next week! (12/10~12/12)
- Use this time to prepare for the final exam on Friday. (12/13)

Objectives

- Pointer to pointer and multidimensional arrays
- Matrix multiplication example

Three ways to define a 2D array

```
int array1[10][10];  
int (*array2)[10];  
int **array3;
```

1. 2D array of size 10x10 is defined

2. Pointer to the first element in the 2D array is defined

3. Pointer to the pointer to the first element in the 2D array is defined

Three ways to define a 2D array

```
int array1[10][10];  
int (*array2)[10];  
int **array3;
```

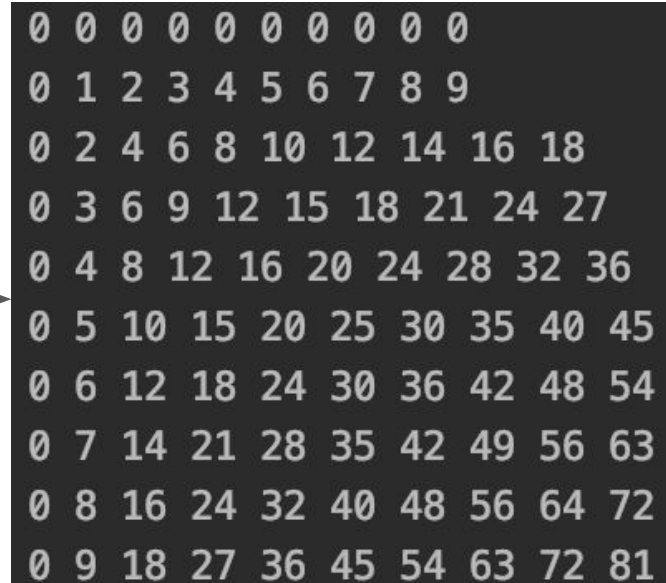
The size of the inner array is known at compile time

The size of the inner array is **NOT** known at compile time

Three ways to define a 2D array

- Method 1
 - We know the dimensions of the array already.
 - The array is defined, and all the slots in the array are allocated in the memory.

```
int array1[10][10];  
  
for (int i = 0; i < 10; i++) {  
    for (int j = 0; j < 10; j++) {  
        array1[i][j] = i*j;  
    }  
}
```



0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9
0	2	4	6	8	10	12	14	16	18
0	3	6	9	12	15	18	21	24	27
0	4	8	12	16	20	24	28	32	36
0	5	10	15	20	25	30	35	40	45
0	6	12	18	24	30	36	42	48	54
0	7	14	21	28	35	42	49	56	63
0	8	16	24	32	40	48	56	64	72
0	9	18	27	36	45	54	63	72	81

Three ways to define a 2D array

- Method 2
 - We know the size of the inner array already.
 - We need to specify how many inner arrays we are going to assign to this 2D array.

```
int (*array2)[10];
```

```
array2 = new int[10][10];
```

```
for (int i = 0; i < 10; i++) {  
    for (int j = 0; j < 10; j++) {  
        array2[i][j] = i*j;  
    }  
}
```

this is fixed

can be of
arbitrary size

0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9
0	2	4	6	8	10	12	14	16	18
0	3	6	9	12	15	18	21	24	27
0	4	8	12	16	20	24	28	32	36
0	5	10	15	20	25	30	35	40	45
0	6	12	18	24	30	36	42	48	54
0	7	14	21	28	35	42	49	56	63
0	8	16	24	32	40	48	56	64	72
0	9	18	27	36	45	54	63	72	81

Three ways to define a 2D array

- Method 2

- This is the same in essence as:

```
int *array;  
array = new int[10];
```

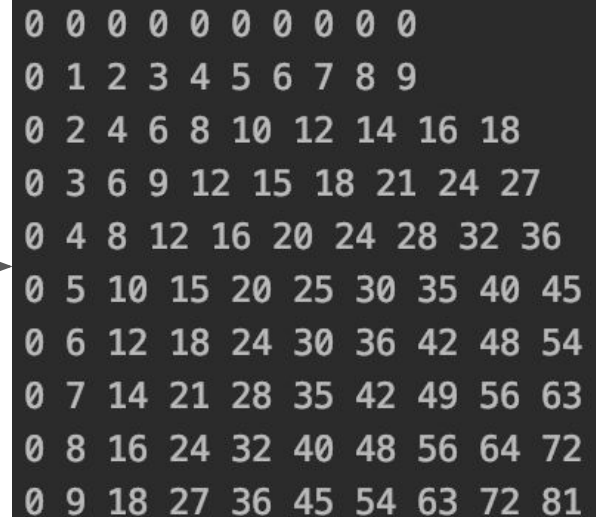
- We need to free the memory used up by 'new'.

```
delete[] array2;
```


Three ways to define a 2D array

- Method 3
 - We know nothing about the size of the array.
 - We need to initialize an array of pointers whose element points to an integer array.

```
int **array3;  
  
array3 = new int*[10];  
  
for (int i = 0; i < 10; i++) {  
    array3[i] = new int[10];  
    for (int j = 0; j < 10; j++) {  
        array3[i][j] = i*j;  
    }  
}
```



0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9
0	2	4	6	8	10	12	14	16	18
0	3	6	9	12	15	18	21	24	27
0	4	8	12	16	20	24	28	32	36
0	5	10	15	20	25	30	35	40	45
0	6	12	18	24	30	36	42	48	54
0	7	14	21	28	35	42	49	56	63
0	8	16	24	32	40	48	56	64	72
0	9	18	27	36	45	54	63	72	81

Three ways to define a 2D array

- Method 3
 - Free all the spaces taken up by 'new'.

```
for (int i = 0; i < 10; i++) {  
    delete array3[i];  
}  
  
delete array3;
```

Matrix Multiplication Example

- Write a program that does the following:
 - generates a random square matrix of size $n \times n$.
 - Takes a matrix and fill it up with integers between 0 and 9.
 - multiplies a square matrix by itself.
 - The result should be stored in the original matrix.
 - calculates the elapsed time of the above two functions.
 - This function takes one square matrix of size $n \times n$ and one of the above function, executes that function with the given matrix, and returns the elapsed time in seconds (double).

Matrix Multiplication

- Vary the square matrix size n to see how much these two functions take.
- Functions to be used are defined in “lab_11.h”.

```
// matrix manipulating functions
void generate_random_mat(int **m, int n);
void matrix_mult(int **m, int n);

// performance measurement function
duration<double> clock(int **m, int n, void (*func)(int **, int));

// utility function
void print_matrix(int **m, int n);
```