CS 161
Introduction to CS I
Lecture 19

- Multidimensional arrays

Oregon State University
College of Engineering

Review static 1D arrays

```
1. const int n_people = 5;
2. int height[n_people];
3. for (int i=0; i<n_people; i++)
4.    height[i] = rand()%13 + 60;
```

See lec19-max-in-array.cpp

- Let's find the tallest person (which index?)

```
1. int tallest = 0; /* start with first person */
2. for (int i=1; i<n_people; i++)
3.    if (height[i] > height[tallest])
4.       tallest = i;
5. cout << "Tallest person: index " << tallest
6.      << " (" << height[tallest] << " inches)" << endl;
```

Review dynamic arrays (on the heap)

- Dynamic array (e.g., when size could change)

```
1. float* g = new float[3];  /* from heap */
2. . . .
3. delete [] g; /* free the memory */
4. g = NULL;
```

- Tip: anytime you use **new**, immediately decide where a **delete** should go
  - Anytime you use **new []**, decide where **delete []** should go

Visualize dynamic arrays

See lec19-dynamic-arrays.cpp

- A farmer plants some number of corn fields each year
  - Different number each year
- Goal: store corn yield (number of bushels) for each field
- We can use the same variable (pointer) each year
  - Allocate the number of fields needed
  - Free them at the end of the year

Your turn: Create an array in a function

- Fill in the blanks:

```
1. int* create_arr(int n) {
2.     _____ new_arr = new _____[___];
3.     return _______;
4. }
```

Oregon State University
College of Engineering

Your turn: Create an array in a function

- Fill in the blanks:

```
1. int* create_arr(int n) {
2.     _____ new_arr = new _____[___];
3.     return _______;
4. }
```

- Solution:

```
1. int* create_arr(int n) {
2.     int* new_arr = new int[n];
3.     return new_arr;
4. }
```

Oregon State University
College of Engineering

Your turn: Initialize array values in a function

- Fill in the blanks to initialize each value to its index * 2:

```
1. ____ init_arr(int* arr, int n) {
2.     for (int i=___; i<___; i++)
3.         arr[___] = _______;
4. }
```

Oregon State University
College of Engineering

Your turn: Initialize array values in a function

- Fill in the blanks to initialize each value to its index * 2:

```
1. ____ init_arr(int* arr, int n) {
2.     for (int i=___; i<___; i++)
3.         arr[___] = _______;
4. }
```

- Solution:

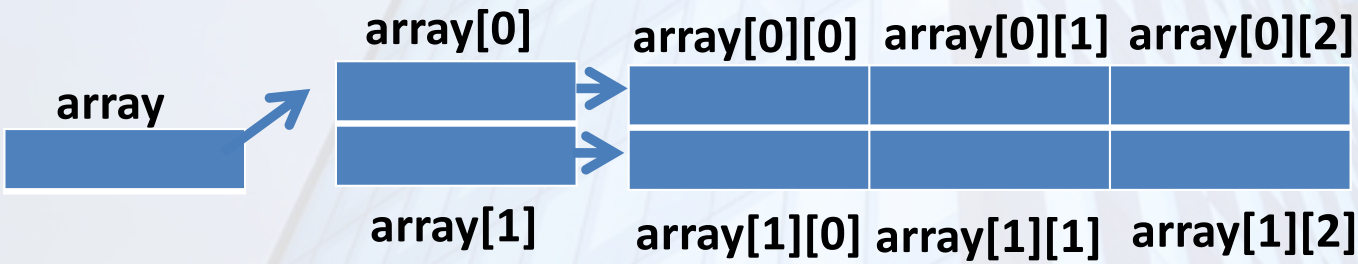
```
1. void init_arr(int* arr, int n) {
2.     for (int i=0; i<n; i++)
3.         arr[i] = i * 2;
4. }
```

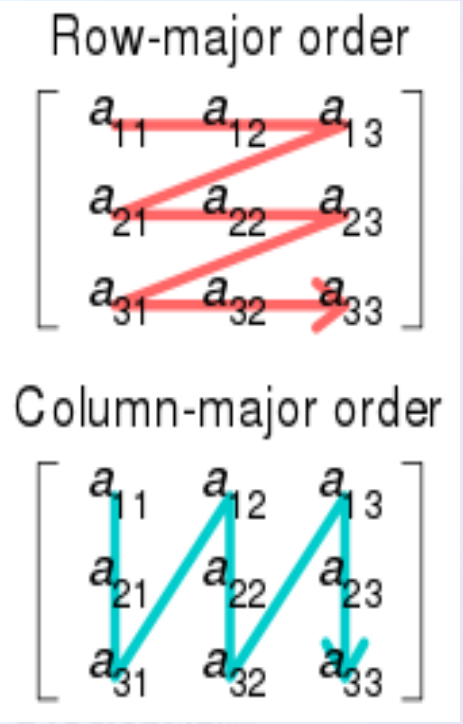
Your ideas for 2D arrays

- Windows in a building
- Rooms on a floor of a building
- Tic-tac-toe board
- Battleship game board
- Keys on a keyboard
- Holes in a waffle
- Many more!

Two-dimensional arrays

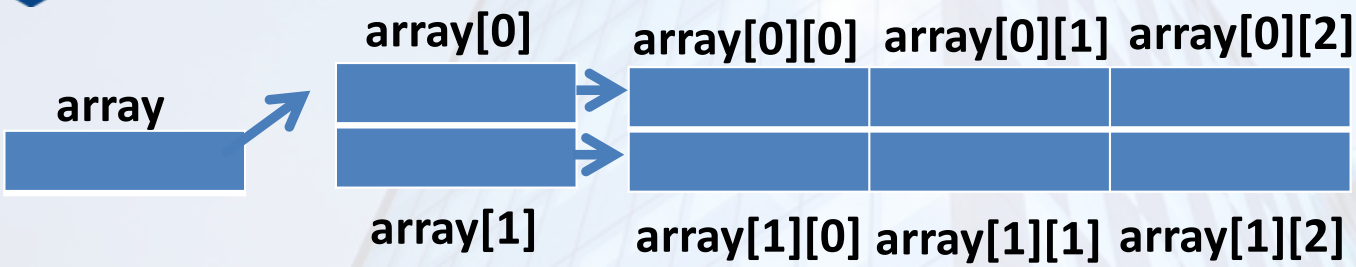
- Array of arrays





- Stride: number of items/locations between successive array elements
- C++ uses row-major order for 2D arrays

Create 2D arrays: pointers to pointers



- Stack

```
1. int stack_arr[2][3];
```

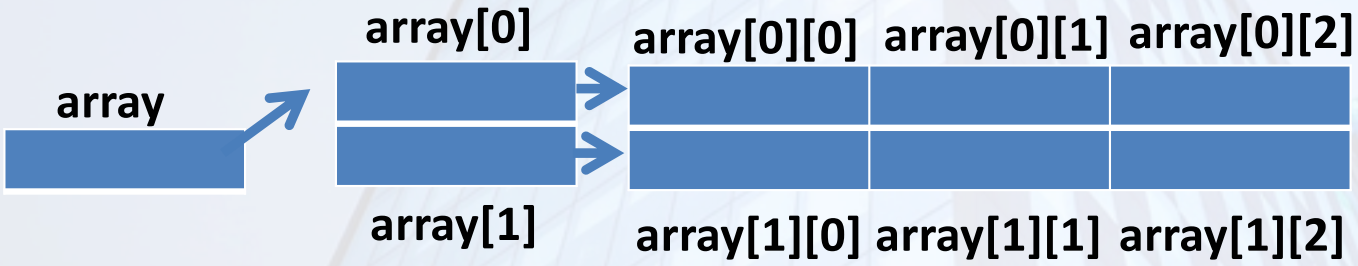
See lec19-2d-arrays.cpp

- Heap

```
1. int** heap_arr = new int*[2];
2. for (int i=0; i<2; i++)
3.   heap_arr[i] = new int[3];
```

Two "new"s

Free 2D arrays on the heap




```
1. for (int i=0; i<2; i++) {
2.   delete [] heap_arr[i];
3.   heap_arr[i] = NULL;
4. }
5. delete [] heap_arr;
6. heap_arr = NULL;
```

See lec19-2d-arrays.cpp

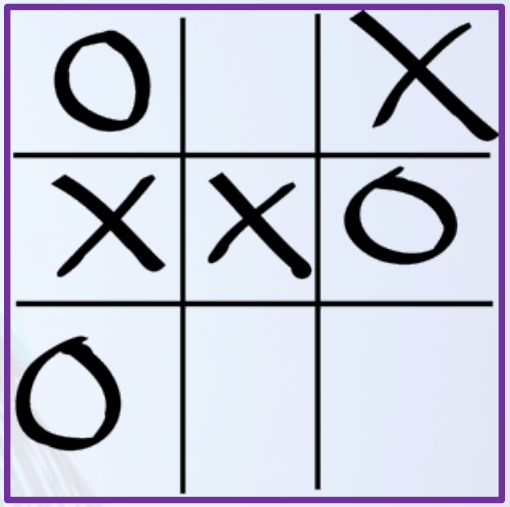
Two "delete"s

Oregon State University
College of Engineering

Your turn: Fill in the blanks to…

- Declare a 2D array of characters to hold a tic-tac-toe board (3x3)

```
1. ____ tic_tac_toe = new ____[___];
2. for (int i=0; i<___; i++)
3.   tic_tac_toe[i] = new ____[___];
```

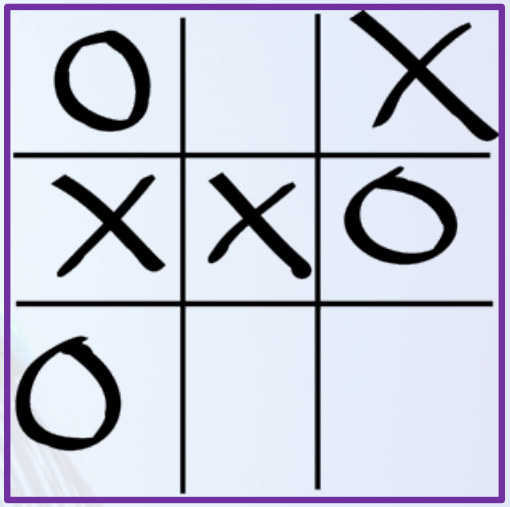


Oregon State University
College of Engineering

Your turn: Fill in the blanks to…

- Declare a 2D array of characters to hold a tic-tac-toe board (3x3)

```
1. ____ tic_tac_toe = new ____[___];
2. for (int i=0; i<___; i++)
3.   tic_tac_toe[i] = new ____[___];
```



- Solution:

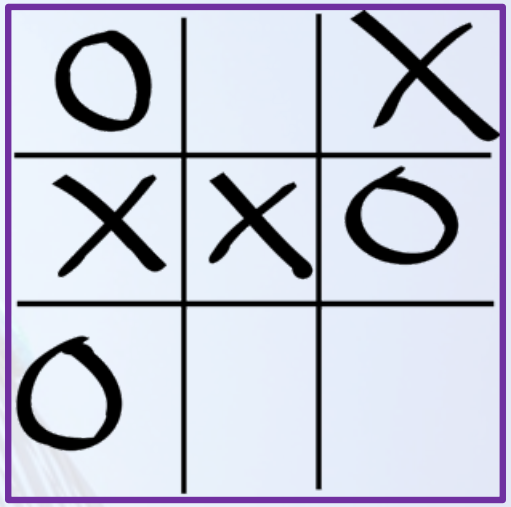
```
1. char** tic_tac_toe = new char*[3];
2. for (int i=0; i<3; i++)
3.   tic_tac_toe[i] = new char[3];
```

Oregon State University
College of Engineering

Your turn: Fill in the blanks to…

- Assign items in your array to 'X' so that you get 3 'X's in a row (any direction)

```
1. tic_tac_toe[____][____] = 'X';
2. tic_tac_toe[____][____] = 'X';
3. tic_tac_toe[____][____] = 'X';
```



- Print out your tic-tac-toe board

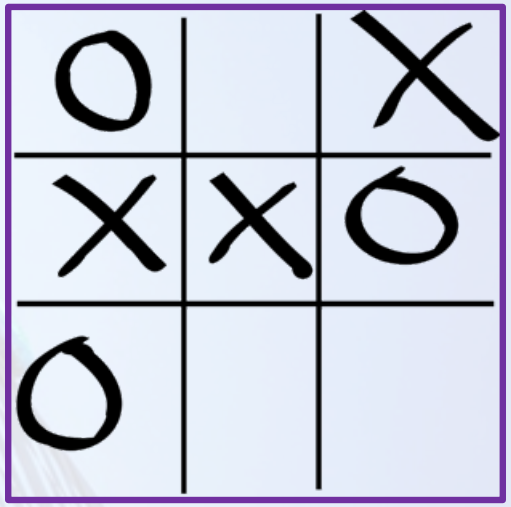
```
1. for (int i=0; i<___; i++) {
2.     for (int j=0; j<___; j++)
3.         cout << tic_tac_toe[___][___];
4.     cout << endl;
5. }
```

Oregon State University
College of Engineering

Your turn: Fill in the blanks to…

- Assign items in your array to 'X' so that you get 3 'X's in a row (any direction)

```
1. tic_tac_toe[____][____] = 'X';
2. tic_tac_toe[____][____] = 'X';
3. tic_tac_toe[____][____] = 'X';
```



- Print out your tic-tac-toe board

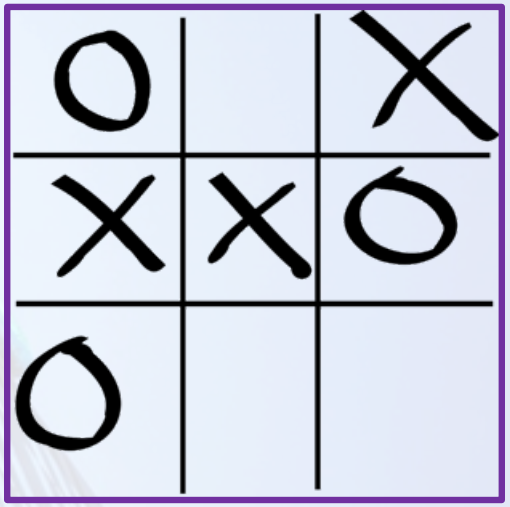
```
1. for (int i=0; i<3; i++) {
2.     for (int j=0; j<3; j++)
3.         cout << tic_tac_toe[i][j];
4.     cout << endl;
5. }
```

Oregon State University
College of Engineering

Your turn: Fill in the blanks to…

- Free your tic-tac-toe board

```
1. for (int i=0; i<___; i++)
2.    delete __________;
3. delete _________;
```



Oregon State University
College of Engineering

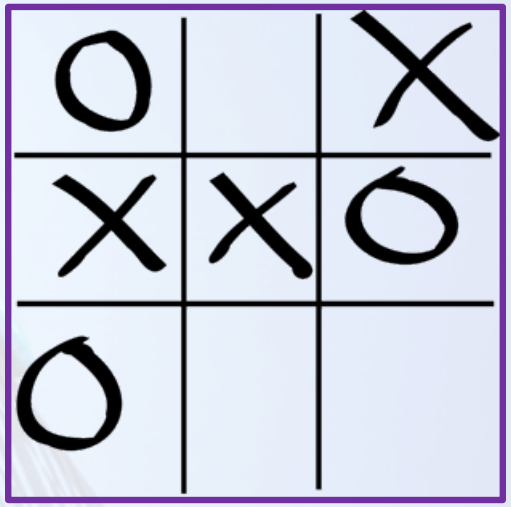
Your turn: Fill in the blanks to…

- Free your tic-tac-toe board

```
1. for (int i=0; i<___; i++)
2.     delete __________;
3. delete _________;
```

- Solution:

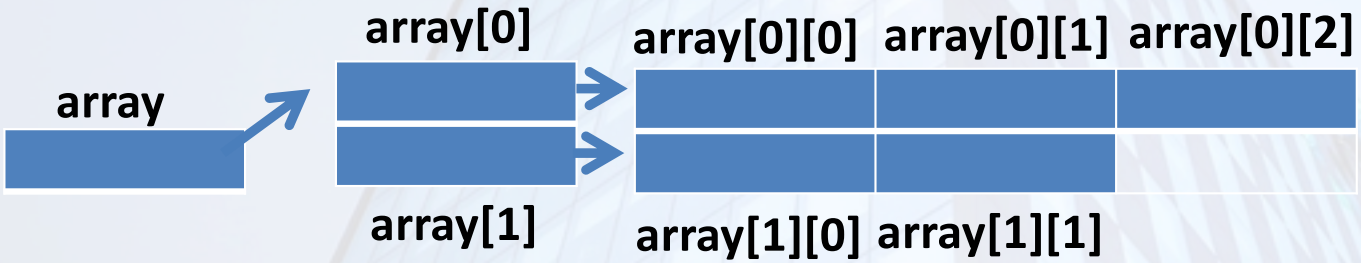
```
1. for (int i=0; i<3; i++)
2.     delete [] tic_tac_toe[i];
3. delete [] tic_tac_toe;
```



See lec19-tictactoe.cpp

Oregon State University
College of Engineering

Jagged arrays




```
1. int** array = new int*[2];
2. array[0] = new int[3];
3. array[1] = new int[2];
```

How would you free this heap memory?

Same way as with rectangular 2D array.

Oregon State University
College of Engineering

Passing static 2D arrays to functions

- Static: must include the size of both dimensions, or at least the final dimension

```
1. int main() {
2.    int array[5][5];
3.
4.    pass_2Darray(array);

5.    return 0;
6. }
```

```
1. void pass_2Darray(int a[5][5]) {
2.    cout << a[0][0] << endl;
3. }
4. /* OR */
5. void pass_2Darray(int a[][5]) {
6.    cout << a[0][0] << endl;
7. }
```

Passing dynamic 2D arrays to functions

- Static: must include the size of both dimensions, or at least the final dimension

```
1. int main() {
2.   int** array;
3.   /* allocate array */
4.   pass_2Darray(array);
5.   /* free array */
6.   return 0;
7. }
```

```
1. void pass_2Darray(int* a[]) {
2.   cout << a[0][0] << endl;
3. }
4. /* OR */
5. void pass_2Darray(int** a) {
6.   cout << a[0][0] << endl;
7. }
```

Oregon State University
College of Engineering

What vocabulary did we learn today?

- Stride
- Row-major order
- Column-major order
- Jagged array

What ideas and skills did we learn today?

- How to declare 2D arrays on the stack
- How to declare (and delete) 2D arrays on the heap
- How 2D arrays are arranged in memory
  - Pointers to pointers
  - Row-major order
- How to pass 2D arrays to functions
  - (More on this next time)

Oregon State University
College of Engineering

Week 7 continues

- ❑ Attend lab (laptop required)
- ❑ Read **Rao Lesson 4** (pp. 71-74)
  Rao Lesson 6 (pp. 145-146)
- ❑ Study session Thursday 2/20, 6-7 p.m. in LINC 268
- ❑ **Assignment 4** (due **Sunday, Feb. 23**)

See you Friday!

- ❑ Bring: [name of] your favorite collectible item