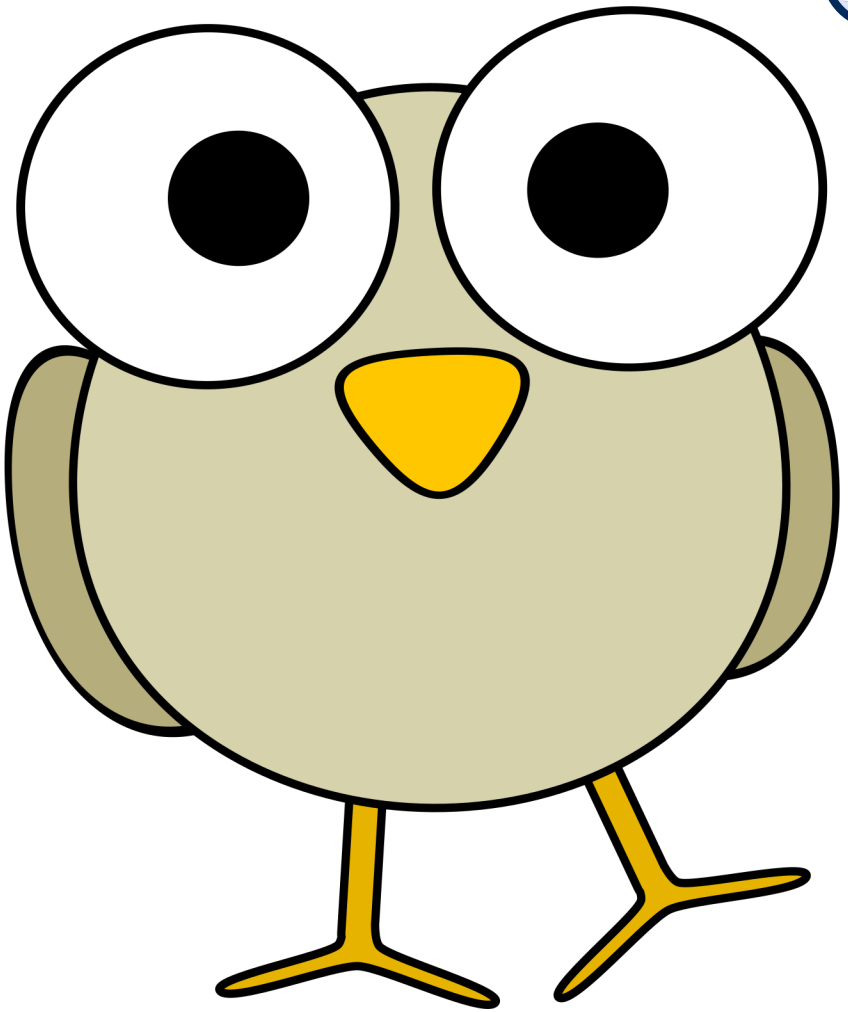
CSCI 200: Foundational Programming Concepts & Design Lecture 39

Multidimensional Lists

Previously in CSCI 200

- Sorting Algorithms
  - Selection Sort $O(n^2)$
  - Insertion Sort $O(n^2)$
  - Bubble Sort $O(n^2)$
  - Merge Sort: $O(n \log n)$
- Searching Algorithms
  - Linear Search: $O(n)$
  - Binary Search: $O(\log n)$
    - List must be sorted first
    - Most efficient to implement iteratively

Questions?



Any Sorting Algorithm

```
void sort(List& list) {
   ...
   if( list[i] < list[j] ) ...
   ...
}
```

In Practice

```
void sort(List<T>& list) {
   ...
   if( T ? T ) ...
   ...
}
```

- ? is some comparison operator

  < > <= >=

Abstract the Comparison!

```
void sort_list(List& list, function compare) {
   ...
   if( compare(list[i], list[j]) ) ...
   ...
}
```

- Pass in a function as a parameter to use!

Function Pointer As A Parameter

```
template<typename T>
void sort_list(List<T> list, bool (*pfCompare)(T, T)) {
   ...
   if( pfCompare(list[i], list[j]) ) ...
   ...
}
```

Function Pointers As An Argument

```
bool ascending(int x, int y) { return x < y; }
bool descending(int x, int y) { return x > y; }

int main() {
  List<int> intList;
  // populate as 5 2 4 6 3 1
  print_list(intList); // prints 5 2 4 6 3 1
  sort_list(intList, ascending);
  print_list(intList); // prints 1 2 3 4 5 6
  sort_list(intList, descending);
  print_list(intList); // prints 6 5 4 3 2 1
  return 0;
}
```

Custom Sorting!

```
bool ascending(int x, int y) { return x < y; }
bool evens_first(int x, int y) {
  if(x%2==0 && y%2==1) return true;
  if(x%2==1 && y&2==0) return false;
  return ascending(x, y);
}
int main() {
  List<int> intList;
  // populate as 5 2 4 6 3 1
  print_list(intList); // prints 5 2 4 6 3 1
  sort_list(intList, ascending);
  print_list(intList); // prints 1 2 3 4 5 6
  sort_list(intList, evens_first);
  print_list(intList); // prints 2 4 6 1 3 5
  return 0;
}
```

Learning Outcomes For Today

- Sketch how a multidimensional list is stored in memory. Create and use multidimensional lists.
- Sketch how BFS and DFS each traverse a multidimensional list.
- Implement BFS and DFS. Explain the uses of a queue and stack in each process.

On Tap For Today

- Multidimensional Lists
- Searching A Grid
  - BFS or DFS
- Practice

On Tap For Today

- Multidimensional Lists
- Searching A Grid
  - BFS or DFS
- Practice

List Element Types

```
List<T> myList;
```

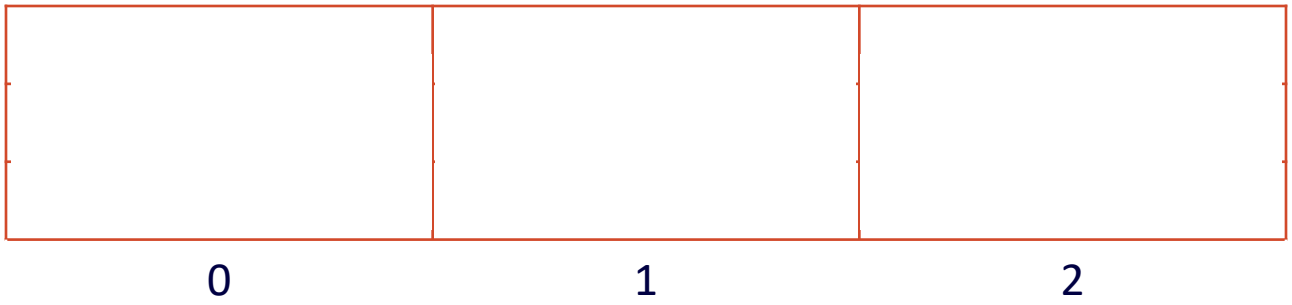
- What can `T` be?

List Element Types

```
List<T> myList;
```

- What can `T` be?
  - `int, bool, char, float, double`
  - `string`
  - `struct`
  - `class`
  - `List<T>` !

Lists Can Hold Lists



Two-Dimensional Lists

'a'	'b'	'c'	'q'	'w'	'e'	'x'	'y'	'z'
0	1	2	0	1	2	0	1	2
	0			1			2	

	0	1	2
0	'a'	'b'	'c'
1	'q'	'w'	'e'
2	'x'	'y'	'z'

LinkedList Implementation

```
LinkedList< LinkedList<T>* > *pMyListOfLists = nullptr;
pMyListOfLists = new LinkedList< LinkedList<T>* >();

pMyListOfLists->insert( 0, new LinkedList<T>() );
pMyListOfLists->insert( 1, new LinkedList<T>() );

pMyListOfLists->at(0)->insert( 0, T() );
pMyListOfLists->at(1)->insert( 0, T() );
pMyListOfLists->at(1)->insert( 1, T() );

T item = pMyListOfLists->at(1)->at(1)
```

Array Implementation

- Declaring Multidimensional Static Arrays
- Declaring Multidimensional Dynamic Arrays
- Accessing Values in Multidimensional Arrays

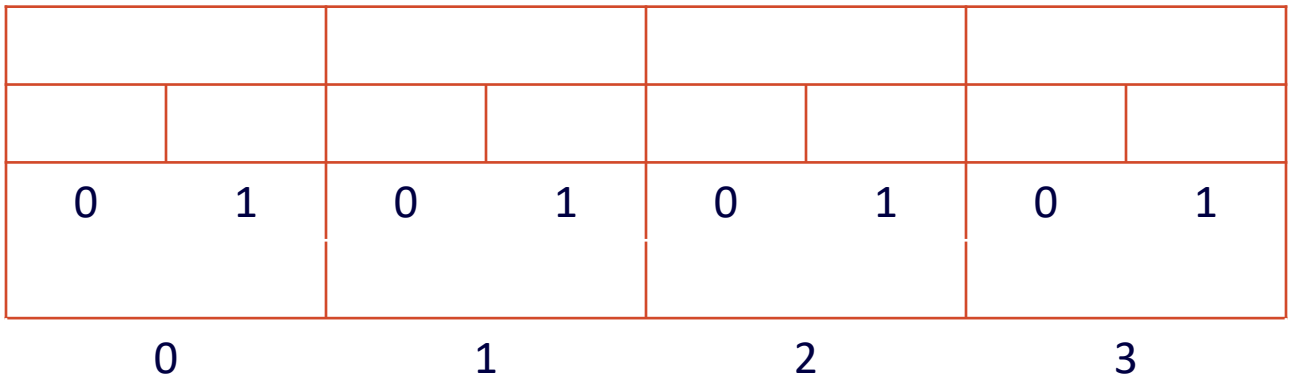
Array Implementation

- Declaring Multidimensional Static Arrays
- Declaring Multidimensional Dynamic Arrays
- Accessing Values in Multidimensional Arrays

Declaring Static 2D Array

- "Computer, create an array that can hold 4 elements. Inside each element create an array that can hold 2 elements."

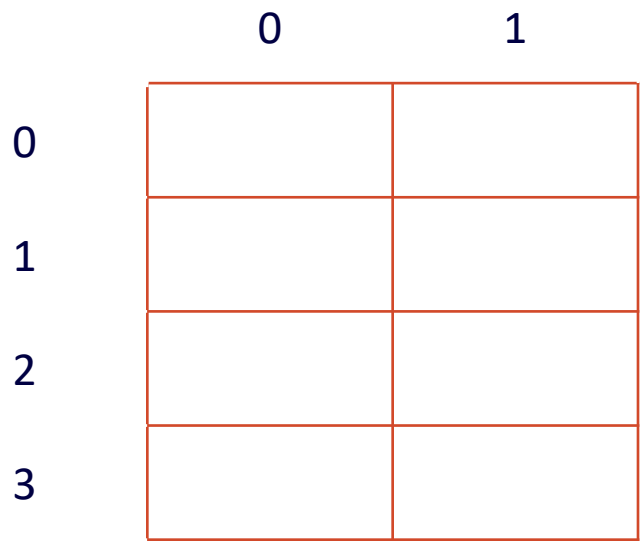
```
char myArray[4][2];
```



Declaring Static 2D Array

- "Computer, create a table that has 4 rows and 2 columns."

```
int myTable[4][2];
```



Which are valid?

```
int temp[2][5];

bool seatsFilled[8][8];

double prices[2][];

char vowels[1][5];

char vowels2[5][1];

int array[][5];
```

Which are valid?

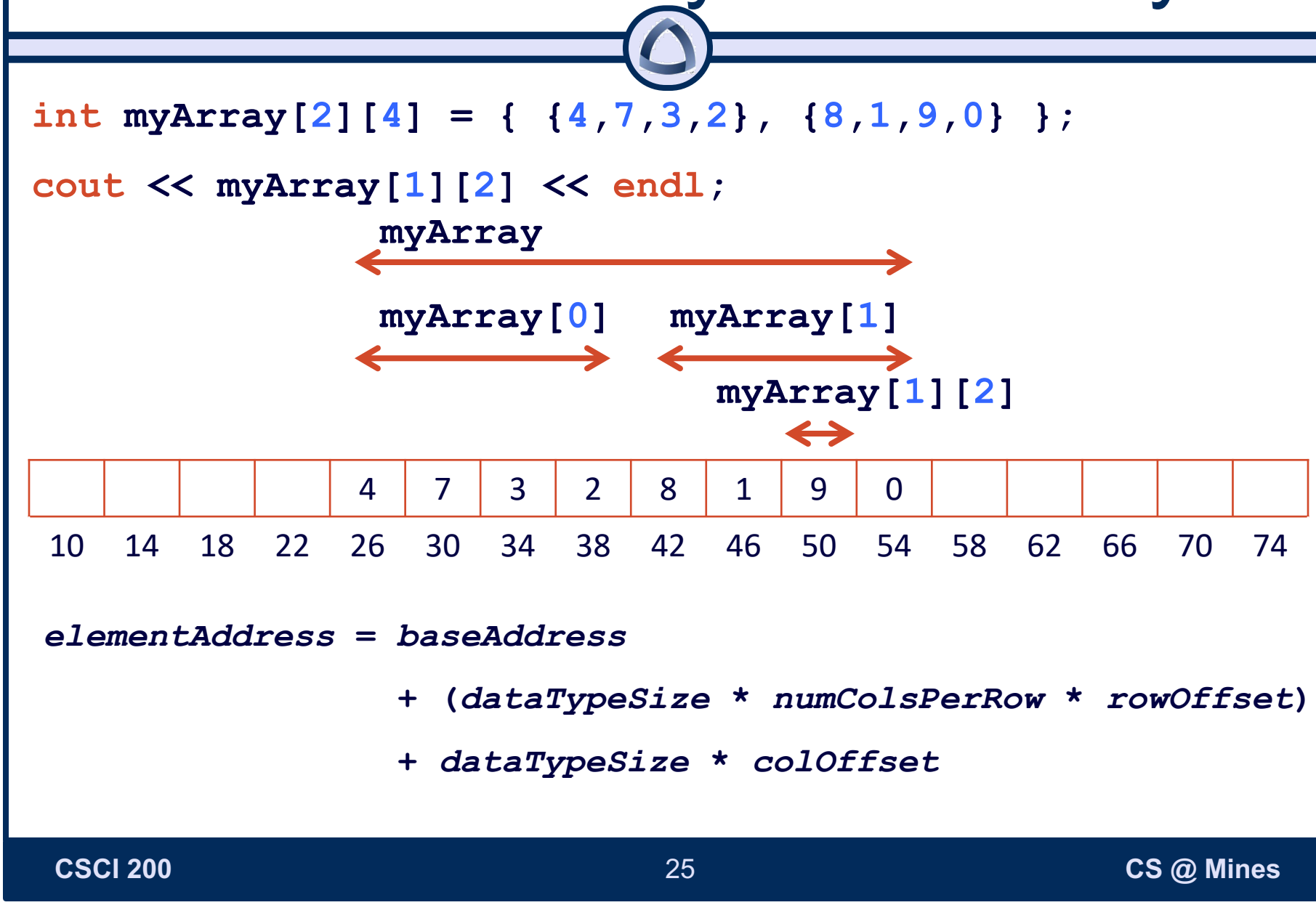
```
int temp[2][5];

bool seatsFilled[8][8];

double prices[2][];          // no cols specified

char vowels[1][5];

char vowels2[5][1];

int array[][5];              // no rows specified
```

Initializing Static 2D Array

```
char myArray[4][2] = { {'a', 'b'},
                       {'o', 'k'},
                       {'c', 'd'},
                       {'e', 'f'} };
```

'a'	'b'	'o'	'k'	'c'	'd'	'e'	'f'
0	1	0	1	0	1	0	1
0		1		2		3	

Static 2D Arrays in Memory



```
int myArray[2][4] = { {4,7,3,2}, {8,1,9,0} };

cout << myArray[1][2] << endl;
```

myArray

myArray[0] myArray[1]

myArray[1][2]

				4	7	3	2	8	1	9	0					
10	14	18	22	26	30	34	38	42	46	50	54	58	62	66	70	74

elementAddress = baseAddress

*+ (dataTypeSize * numColsPerRow * rowOffset)*

*+ dataTypeSize * colOffset*

Initialization Practice

- What happens in each case?

```
int array[2][3] = { {1,2,3}, {4,5,6} };

int array2[2][5] = { {1,2,3}, {4,5} };

int array3[][5] = { {1,2}, {3,4,5,6}, {7} } ;

int array4[5][] = { {1,2}, {3,4,5} };

int array5[3][3] = { 1,2,3,4,5,6,7 };
```

Initialization Practice

- What happens in each case?

```
int array[2][3] = { {1,2,3}, {4,5,6} };

int array2[2][5] = { {1,2,3}, {4,5} };

int array3[][5] = { {1,2}, {3,4,5,6}, {7} } ;

int array4[5][] = { {1,2}, {3,4,5} };

int array5[3][3] = { 1,2,3,4,5,6,7 };
```

Array Implementation

- Declaring Multidimensional Static Arrays
- Declaring Multidimensional Dynamic Arrays
- Accessing Values in Multidimensional Arrays

Dynamic Arrays

- **One-Dimensional**

```
int n;
cout << "Enter size: ";
cin >> n;
T* pArray = new T[n];
```

Dynamic Arrays

- **Two-Dimensional**

```
int n1, n2;
cout << "Enter size 1: ";
cin >> n1;
T** pArray = new T*[n1];
for(int i = 0; i < n1; i++) {
   cout << "Enter size 2: ";
   cin >> n2;
   pArray[i] = new T[n2];
}
```

Array Implementation

- Declaring Multidimensional Static Arrays
- Declaring Multidimensional Dynamic Arrays
- Accessing Values in Multidimensional Arrays

Accessing Values

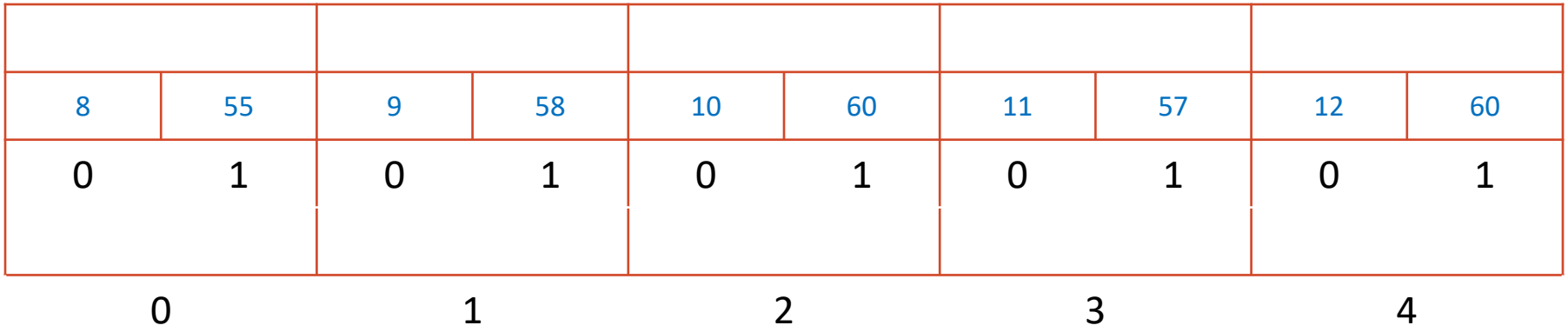
```
int** classSize = ...;
```

8	55	9	58	10	60	11	57	12	60
0	1	0	1	0	1	0	1	0	1
0		1		2		3		4	

How to access the integer 57?

Accessing Values

```
int** classSize = ...;
```



How to access the integer 57? `classSize[3][1];`

Accessing Values

```
int** classSize = ...;
```

	0	1
0	8	55
1	9	58
2	10	60
3	11	57
4	12	60

```
classSize[2][0]; // =
classSize[0][1]; // =
classSize[3];    // =
classSize;       // =
```

Accessing Values

```
int** classSize = ...;
```

	0	1
0	8	55
1	9	58
2	10	60
3	11	57
4	12	60

```
classSize[2][0]; // = 10
classSize[0][1]; // =
classSize[3];    // =
classSize;       // =
```

Static v Dynamic

- What happens in each scenario?
- Scenario 1

```
int my2DArray[4][3] = ...;
my2DArray[1][4] = 9;
```

- Scenario 2

```
int** pMy2DArray = new int*[4];
for(int i = 0; i < 4; i++)
  pMy2DArray[i] = new int[3];
pMy2DArray[1][4] = 9;
```

Higher Dimension Arrays

- Can create 3D and higher dimensional arrays

```
// static
int rubiksCube[3][3][3];

// dynamic
int*** pMegaRubiksCube = new int**[n];
for(int i = 0; i < n; i++) {
  pMegaRubiksCube[i] = new int*[n];
  for(int j = 0; j < n; j++) {
    pMegaRubiksCube[i][j] = new int[n];
  }
}
```

For Funzies

```
int studentLoc[8][7][54][51][56][76][4][50][60][2][2][3];
```

Dimensions refer to:

1. Planet student is on
2. Continent student is in
3. Country student is in
4. State student is in
5. College student is at
6. Building student is in
7. Floor student is on
8. Room student is in
9. Seat student is in
10. Monitor student is looking at
11. Pixel student is looking at
12. RGB color of pixel

Array:

- has 93,518,337,536,000 integer elements!!
- is 2.99e15 bytes large

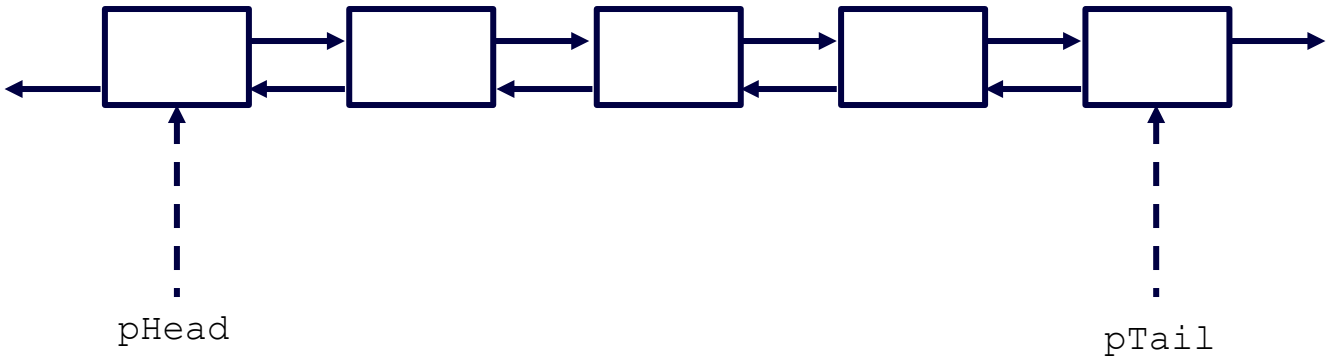
  == 2.66 PB (petabytes!) large

On Tap For Today

- Multidimensional Lists
- Searching A Grid
  - BFS or DFS
- Practice

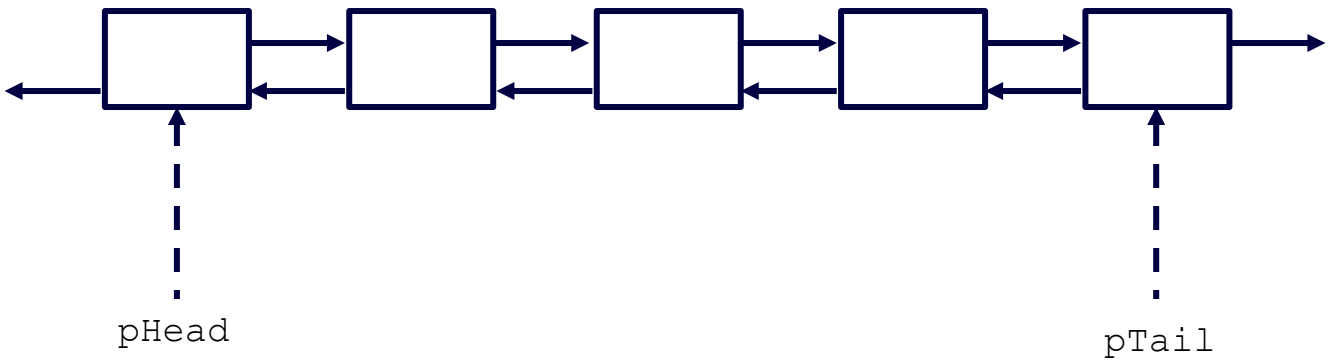
List Operations

Data Structure	push Front()	pop Front()	push Back()	pop Back()	insert Middle()	remove Middle()	traverse Forward()	traverse Backward()
Singly Linked List	✓	✓	✓	✓	✓	✓	✓	✓
Doubly Linked List	✓	✓	✓	✓	✓	✓	✓	✓



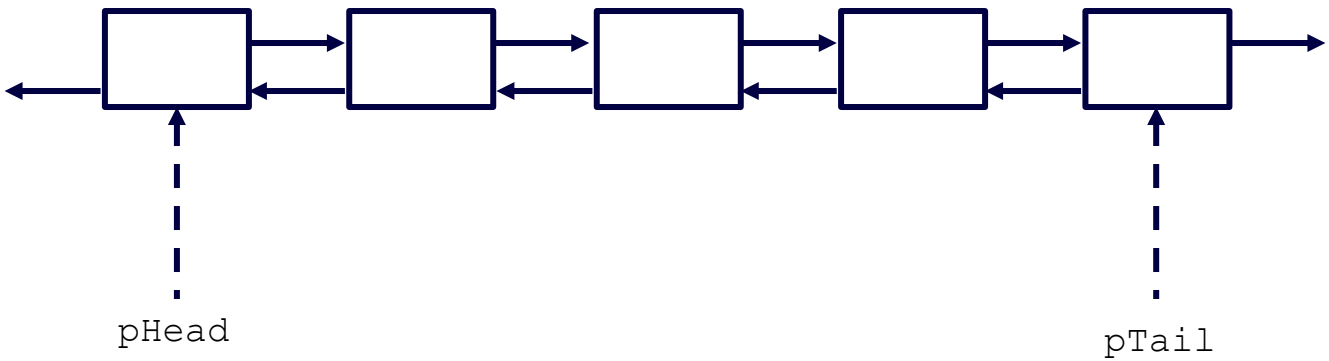
One-dimensional Lists

- Options to search for target?



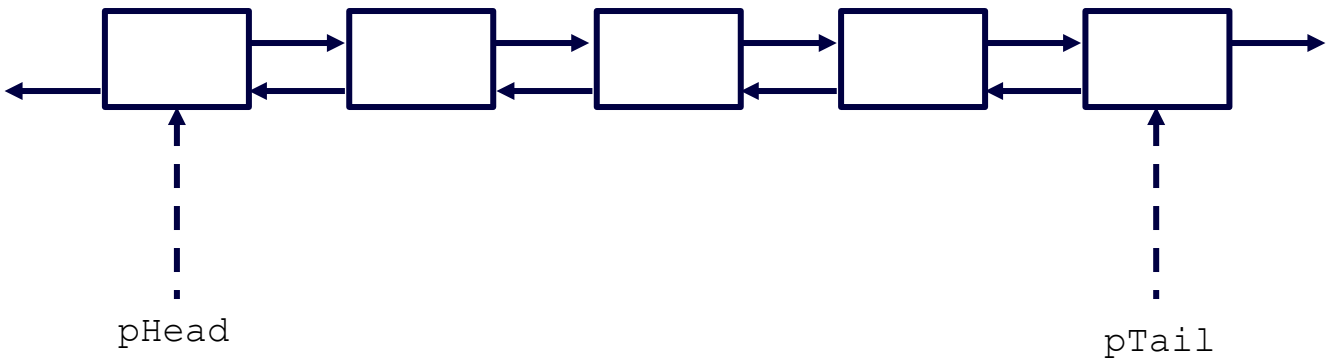
One-dimensional Lists

- Options to search for target?
  - Linear



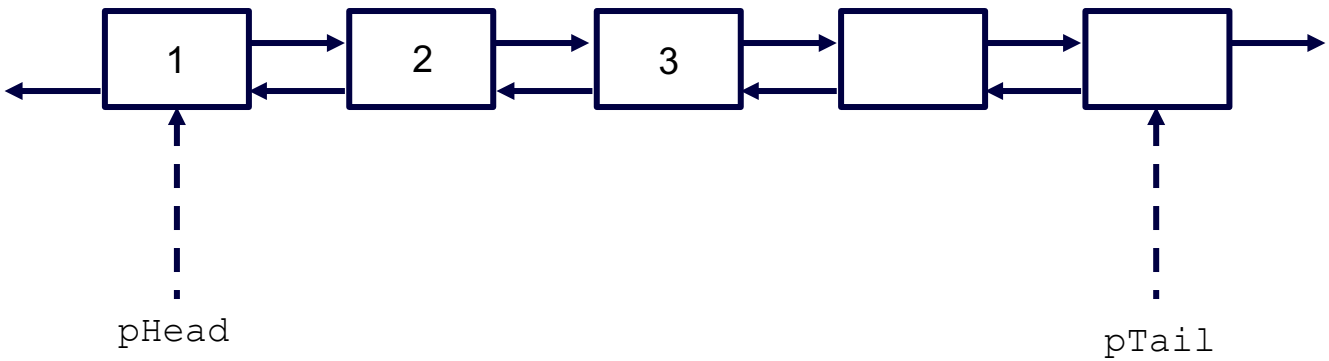
One-dimensional Lists

- Options to search for target?
  - Linear: where to start? where to go next?



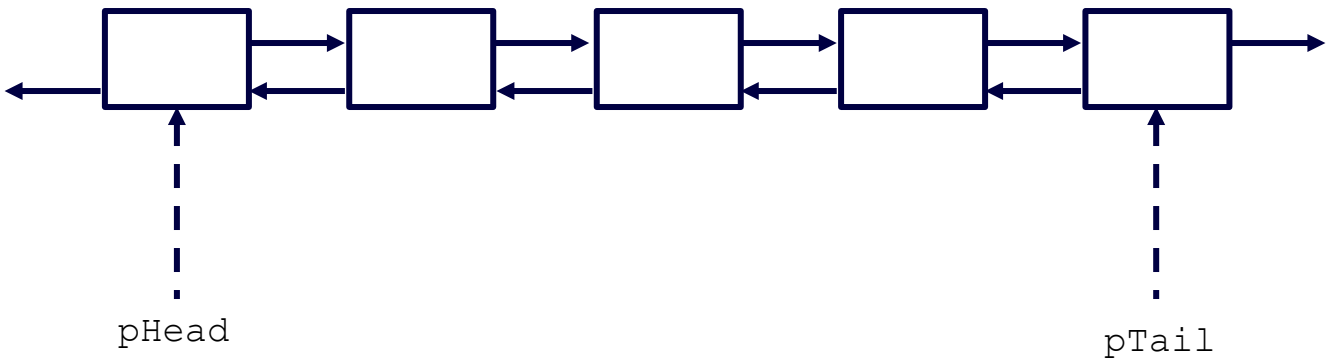
One-dimensional Lists

- Options to search for target?
  - Linear



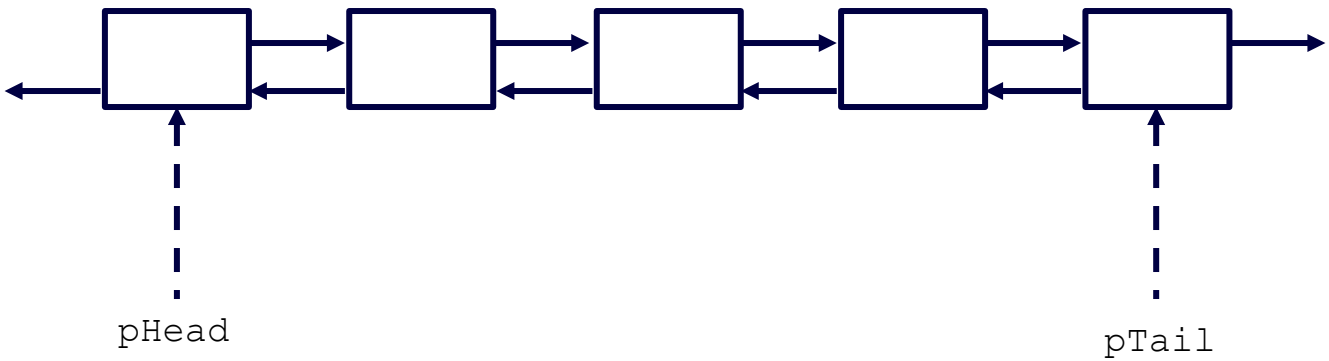
One-dimensional Lists

- Options to search for target?
  - Linear
  - Binary



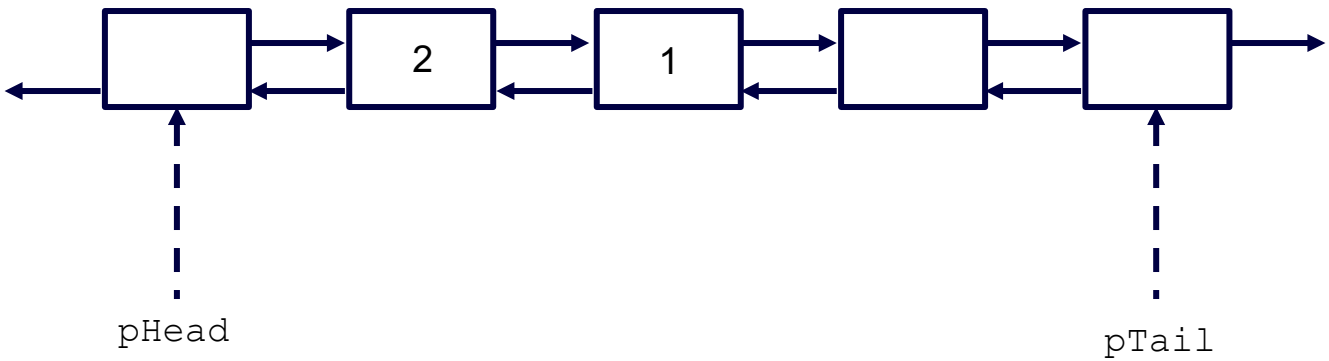
One-dimensional Lists

- Options to search for target?
  - Linear
  - Binary: where to start? where to go next?



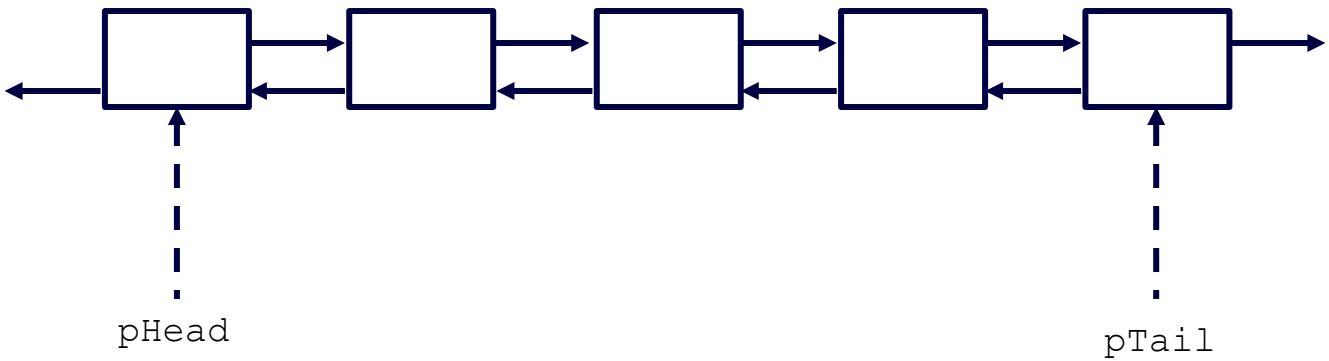
One-dimensional Lists

- Options to search for target?
  - Linear
  - Binary



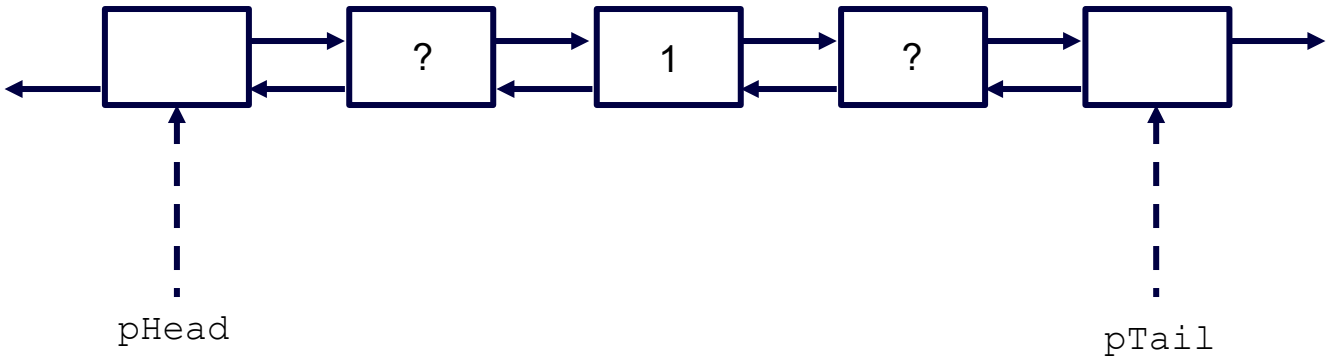
One-dimensional Lists

- Options to search for target?
  - Linear
  - Binary
- Consider an “unsorted-binary search”
  - Where to start? Where to go next?



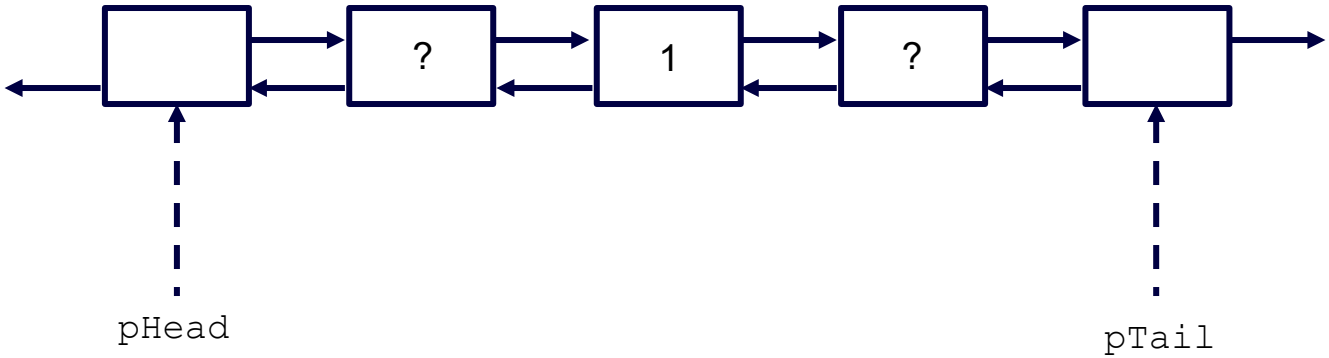
One-dimensional Lists

- Options to search for target?
  - Linear
  - Binary
- Consider an "unsorted-binary search"
  - Where to start? Where to go next?



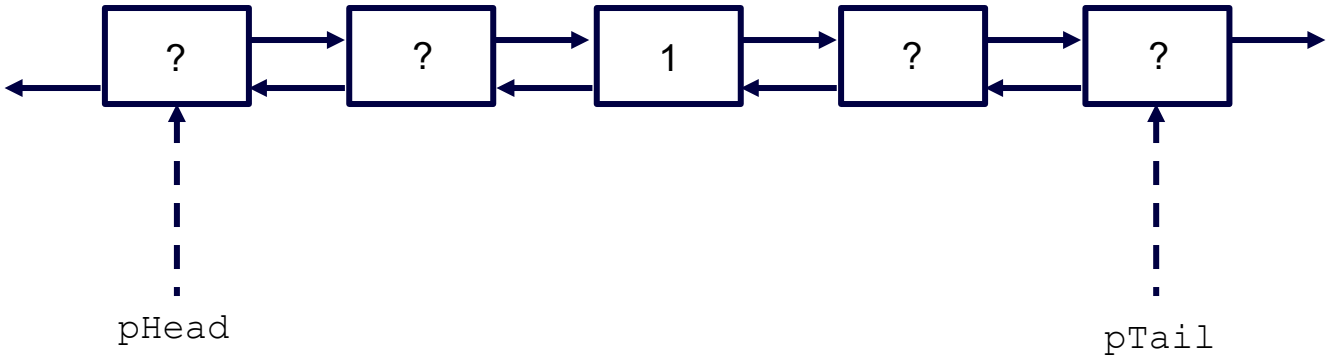
One-dimensional Lists

- Options to search for target?
  - Linear
  - Binary
- Consider an "unsorted-binary search"
  - Where to start? Where to go next? Which to do first?

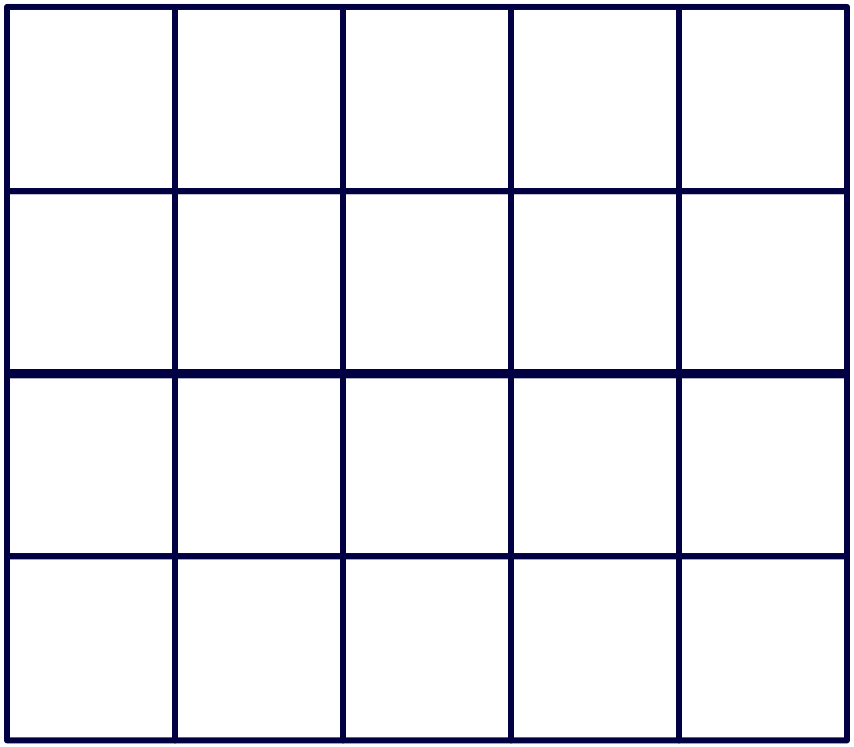


One-dimensional Lists

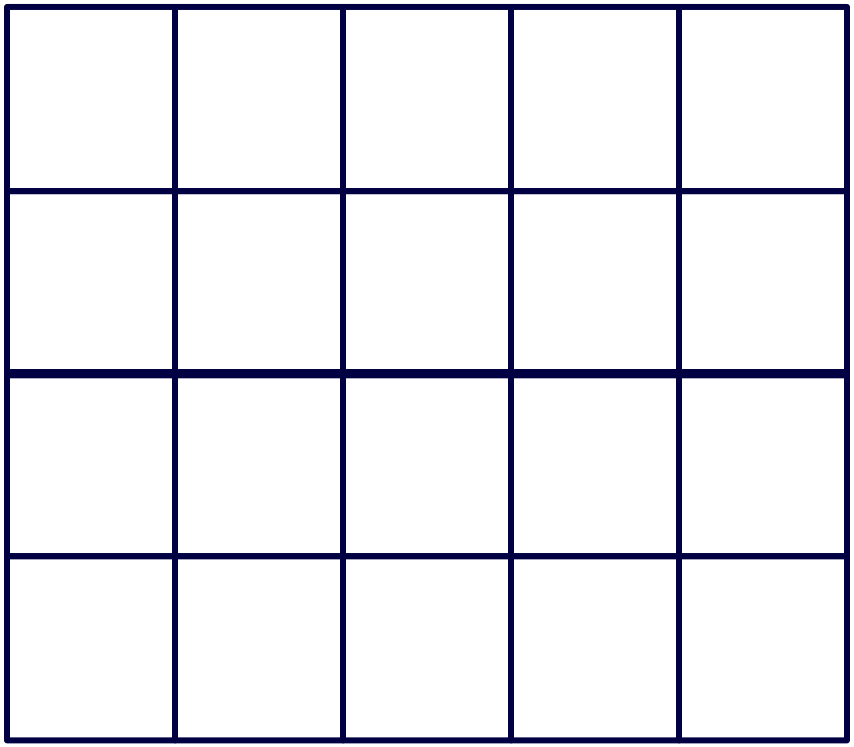
- Options to search for target?
  - Linear
  - Binary
- Consider an "unsorted-binary search"
  - Where to start? Where to go next? Which to do first? Then what?



Multidimensional Lists



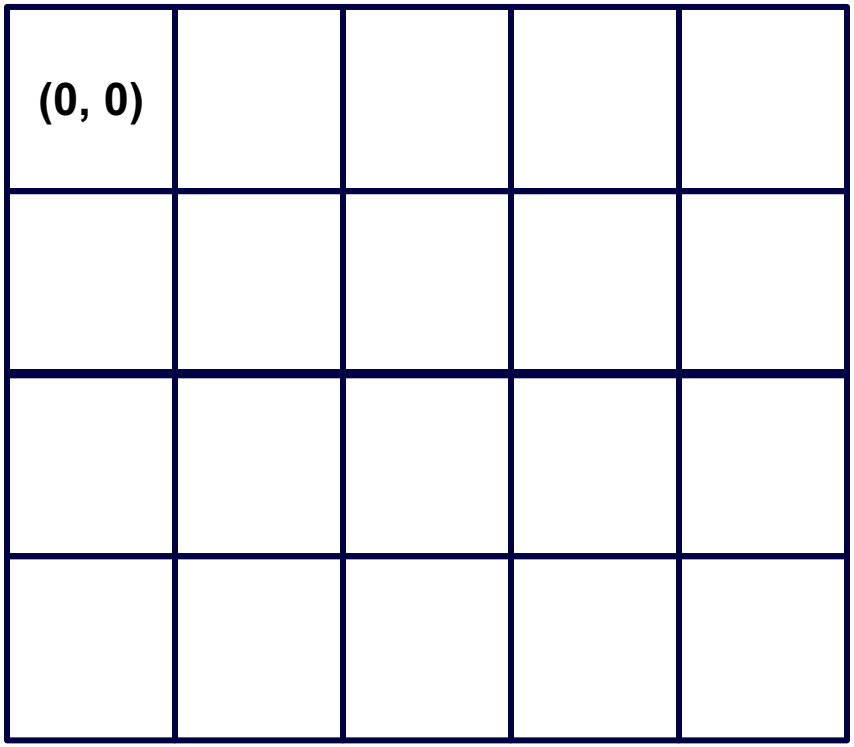
Where To Start?



2D List Number

(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)
(1, 0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)
(2, 0)	(2, 1)	(2, 2)	(2, 3)	(2, 4)
(3, 0)	(3, 1)	(3, 2)	(3, 3)	(3, 4)

Where To Start?



Where To Go Next?

(0, 0)	?			
?				

Then Where?

(0, 0)	?	?		
?	?			
?				

On Tap For Today

- Multidimensional Lists
- Searching A Grid
  - BFS or DFS
- Practice

Multidimensional Searches

- Breadth-First Search (BFS)
  - Search all neighbors first, then search neighbors of neighbors, and so forth
- Depth-First Search (DFS)
  - Search one direction first, then backtrack and search a different direction, and so forth

Example BFS Search Ordering

(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)
(1, 0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)
(2, 0)	(2, 1)	(2, 2)	(2, 3)	(2, 4)
(3, 0)	(3, 1)	(3, 2)	(3, 3)	(3, 4)

Example BFS Search Ordering

0	2	5	9	13
1	4	8	12	16
3	7	11	15	18
6	10	14	17	19

Example DFS Search Ordering

(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)
(1, 0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)
(2, 0)	(2, 1)	(2, 2)	(2, 3)	(2, 4)
(3, 0)	(3, 1)	(3, 2)	(3, 3)	(3, 4)

Example DFS Search Ordering

0	17	16	11	10
1	18	15	12	9
2	19	14	13	8
3	4	5	6	7

Multidimensional Search Pseudocode

```
create list of positions to check

initial list is start node position
mark start node as visited

while there are still nodes to check
        get current node to check
        check if current node is target
        if yes, found!
        if no,
                for each neighbor
                        if neighbor exists and is unvisited
                                add neighbor to list to check
                                mark neighbor as visited
```

Two Questions

1. How to mark node as visited?
2. How to store and process nodes to visit?

1. Tracking Visited Nodes

- Create a second multidimensional list of Booleans
  - if booleanTable[i][j] == true
    - dataTable[i][j] has been visited

How To Determine Next Node?

On Tap For Today

- Multidimensional Lists
- Searching A Grid
  - BFS or DFS
- Practice

A6 Tasks

- 2D Array of characters represents a maze

```
##########################
#S#............#....#....#
#.#...###....###.##.#.##E#
#.#.###.#..#...#.##.#.##.#
#.....#...###....#....##.#
###.##..##.#..####.##..###
###..##....####....#######
##########################
```

- Read maze in from file, store in 2D dynamic array

A6 Tasks

- 2D Array represents a maze
  - Start at S, search for E

```
############################
#S#............#....#....#
#.#...###....###.##.#.##E#
#.#.###.#..#...#.##.#.##.#
#.....#...###....#....##.#
###.##..##.#..####.##..###
###..##....####....#######
############################
```

A6 Tasks

- 2D Array represents a maze
  - Can only move on . spaces

```
############################
#S#............#....#....#
#.#...###....###.##.#.##E#
#.#.###.#..#...#.##.#.##.#
#.....#...###....#....##.#
###.##..##.#..####.##..###
###..##....####....#######
############################
```

A6 Tasks

- 2D Array represents a maze
  - Shortest Path Solution

```
############################
#S#..vvvvvvvv..#vvvv#vvvv#
#v#vvv###...v###v##v#v##E#
#v#v###.#..#vv.#v##v#v##.#
#vvv..#...###vvvv#.vvv##.#
###.##..##.#..####.##..###
###..##....####....#######
############################
```

A6 Tasks

- 2D Array represents a maze
  - DFS visited nodes

```
############################
#S#..vvvvvvvvvv#vvvv#vvvv#
#v#vvv###...v###v##v#v##E#
#v#v###.#..#vvv#v##v#v##.#
#vvvvv#...###.vvv#.vvv##.#
###v##..##.#..####.##vv###
###vv##....####....#######
############################
```

A6 Tasks

- 2D Array represents a maze
  - BFS visited nodes

```
############################
#S#vvvvvvvvvvvv#vvvv#vvvv#
#v#vvv###vvvv###v##v#v##E#
#v#v###v#vv#vvv#v##v#v##.#
#vvvvv#vv###vvvv#vvvvv##.#
###v##vv##v#vv####v##vv###
###vv##vvvv####vvvv#######
############################
```

A6 Tasks

- 2D Array represents a maze
  - Maze be unsolvable

```
############################
#S#............#....#.##.#
#.#...###....###.##.#.#.E#
#.#.###.#..#...#.##.#.##.#
#.....#...###....#....####
###.##..##.#..####.##.####
###..##....####....#######
############################
```

A6 Tasks

- 2D Array represents a maze
  - Maze be unsolvable

```
############################
#S#............#....#....#
#.#...###....###.##.#.##E#
#.#.###.#..#...#.##.#.##.#
###...#...###....#....##.#
###.##..##.#..####.##..###
###..##....####....#######
############################
```

To Do For Next Time

- Rest of semester
  - F 12/01: Stack & Queue
  - M 12/04: Trees & Graphs, Quiz 6
  - W 12/06: Exam Review
  - R 12/07: Set6, SetXP, Final Project due
  - M 12/11: Final Exam