# 18. 2D Vector Case Study

CPSC 120: Introduction to Programming
Kevin A. Wortman ~ CSU Fullerton

# Agenda

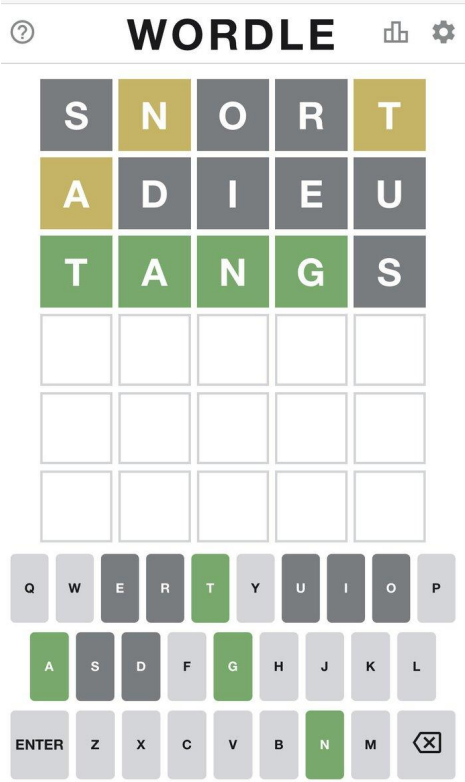
1. Midterm solutions available today
2. Domain: Wordle Hint Frequencies
3. Review: 2D Vectors
4. Live Coding

# 1. Domain: Wordle Hint

# Wordle Hints with Frequencies

- INPUT: a clue string, ex. “tang?”
- OUTPUT: a list of English words that match the clue **and** indicate which match has the highest frequency

# Review: Wordle



# Review: Wordle Clue

- **Clue:** a string containing
  - characters for known letters
  - ? for unknown letters
- Examples
  - “tang?”
  - “vam????”
  - “??oon”

# Review: words.txt

- Unix: /usr/share/dict has **dictionary files**
- **dictionary file:** contains valid words, one word per line
- Start of /usr/share/dict/words:

```
A
A's
AMD
AMD's
AOL
AOL's
AWS
AWS's
Aachen
Aachen's
Aaliyah
Aaliyah's
Aaron
Aaron's
...
```

# Word Frequencies

- **frequency:** how many times something occurs
- Ilya Semenov's frequency data from Wikipedia: https://github.com/IlyaSemenov/wikipedia-word-frequency
- Start of enwiki-2022-08-29.txt:

```
the 183212978
of 86859699
in 75290639
and 74708386
a 53698262
to 52250362
was 32540285
is 23812199
on 21691194
for 21634075
as 21126503
with 18605836
...
```

# Program Requirements

1. Input a **clue** as command line argument; validate
2. Read words from words.txt
3. Make list of English words that match the clue
4. Print the matches
5. (today) **Print out which match is most frequent**

# Example Input/Output

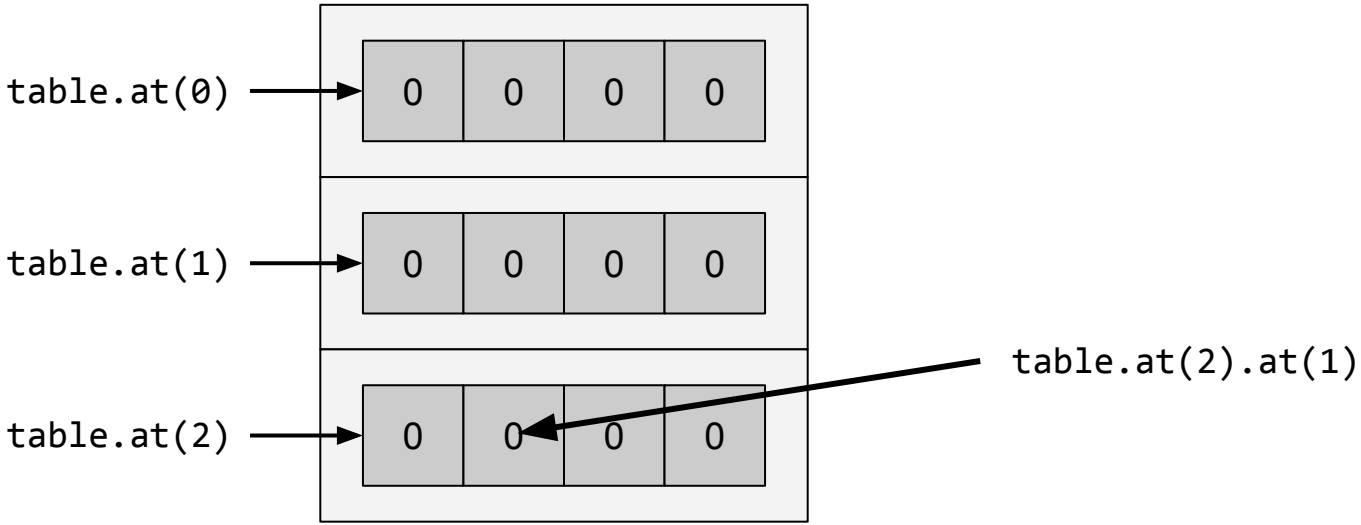
```
$ ./wordle tang?
Your clue matches:
tango
tangs
tangy
The most frequent is: tango

$ ./wordle ??cho
Your clue matches:
Tycho
macho
nacho
The most frequent is: macho
```

# 2. Review: 2D Vectors

# Visualizing a 2D Vector

```
std::vector<std::vector<int>> table(3, std::vector<int>(4, 0));
```



# Initializing a 2D Vector

*statement:*

**std::vector<std::vector<***T***>>** *ident***{**
*rows***, std::vector<***T***>{***cols***,** *value***}}**;

Semantics

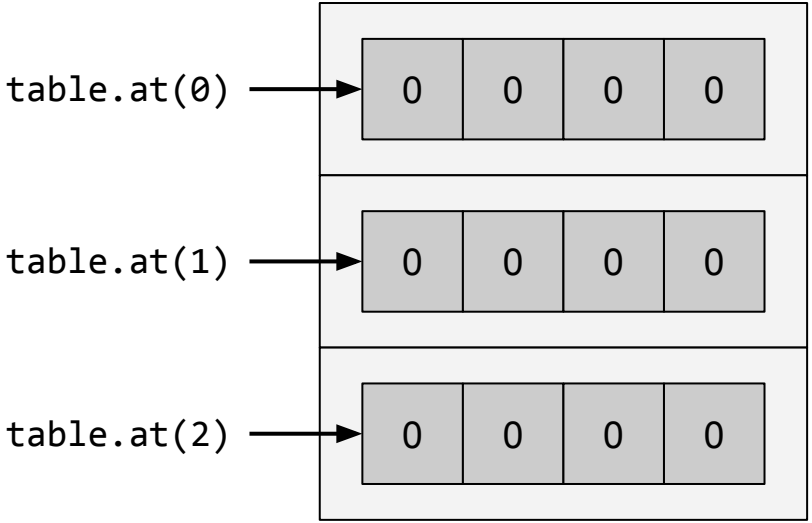
- *ident* is a 2D vector
- dimensions are *rows* and *cols*
- each element is initialized to *value*
- Note: ( ) not { }
- Usually more than 80 characters
  - Style guide says use multiple lines

```
std::vector<std::vector<int>> table(
    3, std::vector<int>(4, 0));
```

# Review: Accessing an Entire Row

- Recall: an element of the outer vector is a row
- Each row is an entire row
- Access a row with `vector::at`, `vector::front`, `vector::back`

```
std::vector<std::vector<int>> table{
    3, std::vector<int>{4, 0}};
```



# 2D Vector Height

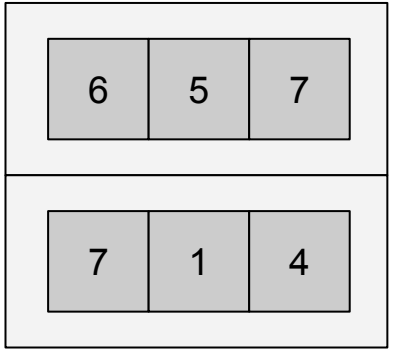
- **Height**
- = number of rows
- = size of outer vector

```
std::vector<std::vector<int>> vec{
  {6, 5, 7},
  {7, 1, 4}
};

std::cout << "height is " << vec.size() << "\n";
```

Output:

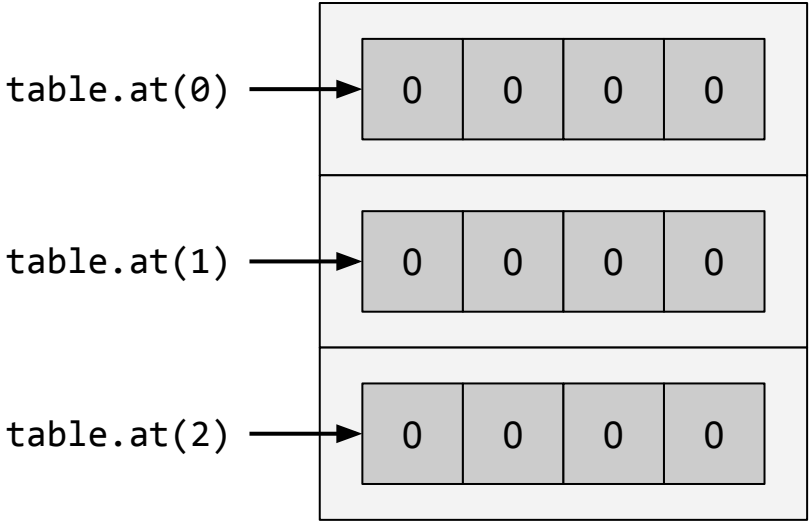
```
height is 2
```



# Accessing an Entire Row

- Recall: an element of the outer vector is a row
- Each row is an entire row
- Access a row with `vector::at`, `vector::front`, `vector::back`

```
std::vector<std::vector<int>> table(
    3, std::vector<int>(4, 0));
```



# 2D Vector Width

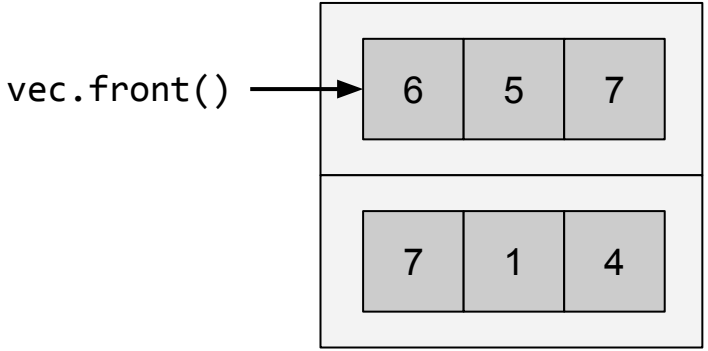
- **Width**
- = number of columns
- = size of any inner vector
  (assuming the vector is not jagged)
- So access a row e.g. `front`
- Get size of that row

```
std::vector<std::vector<int>> vec{
  {6, 5, 7},
  {7, 1, 4}
};

std::cout << "width is " << vec.front().size() << "\n";
```

Output:

```
width is 3
```



# Accessing an Individual Element

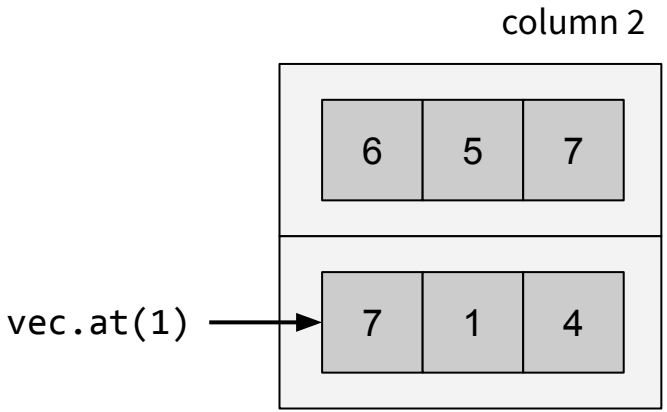
- First: access a row
- Then an element within that row
- Access with `vector::at`, `vector::front`, `vector::back`

```
std::vector<std::vector<int>> vec{
  {6, 5, 7},
  {7, 1, 4}
};

std::cout << vec.at(1).at(2) << "\n";
```

Output:

4



# Iterating Through All Elements

```
// for-each loops

for (const std::vector<int>& row : table) {
   for (const int& cell : row) {
     // use cell, ex.
     std::cout << cell;
   }
 }
```

```
// counter-controlled for loops through all
// indices

for (int i = 0; i < table.size(); ++i) {
  for (int j = 0; j < table.front().size(); ++j) {
    // use table.at(i).at(j) ex.
    std::cout << table.at(i).at(j);
  }
}
```

# Application: Store Entire CSV File

```
std::vector<std::vector<std::string>> ReadCSV(
   const std::string& filename,
   int columns) {
 std::vector<std::vector<std::string>> table;
 std::ifstream file(filename);
```

```
 // read each row
 while (file.good()) {
   std::vector<std::string> row;
   // read each column
   for (int i = 0; i < columns; ++i) {
     std::string cell;
     file.ignore(1, '"'); // leading quote
     std::getline(file, cell, '"');
     file.ignore(1, ','); // comma
     row.push_back(cell);
   }
   if (file.good()) {
     table.push_back(row);
   }
 }

 return table;
}
```

# Application: Store Entire CSV File

```
int main(int argc, char* argv[]) {
 std::vector<std::vector<std::string>> csv{
     ReadCSV("state_demographics.csv", 48)};

 // print states and populations
 bool first{true};
 for (const std::vector<std::string>& row : csv) {
   if (first) {
     first = false;
     continue;
   }
   std::string name{row.at(0)};
   int population{std::stoi(row.at(2))};
   std::cout << name << " population is "
             << population << "\n";
 }

 return 0;
}
```

Output:

```
Connecticut population is 3605944
Delaware population is 989948
District of Columbia population is 689545
Florida population is 21538187
Georgia population is 10711908
Hawaii population is 1455271
Alabama population is 5024279
Alaska population is 733391
Arizona population is 7151502
Arkansas population is 3011524
California population is 39538223
Colorado population is 5773714
...
```

# 3. Live Coding