17. Nested Vectors, CSV

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

Agenda

0. Announce
   a. Reminder: Midterm 2 Wednesday
   b. Sign-in sheet
1. Q&A
2. Nested Loops and Vectors
3. Nested Vector Applications

1. Q&A

Q&A

Let’s hear your questions about...

- This week’s Lab
- Linux
- Any other issues

Reminder: write these questions in your notebook during lab

2. Nested Loops and Vectors

Review: Nesting

- **Nest** (v): put a thing inside the same kind of thing
- Nesting dolls (matryoshka)
- **Nested if**: `if` statement inside `if` statement
- **Nested loop**: loop statement inside loop statement

Nested Loops

Inner loop is inside **outer loop**

```
int number{0};
for (int i = 0; i < 10; i++) {
  for (int j = 0; j < 10; j++) {
    number = number + 2*j - i;
  }
}
```

Recap: Counter-Controlled for Loop

statement:

for (*init-statement***;** *condition***;** *advance-statement* **)**
body-statement

Semantics:

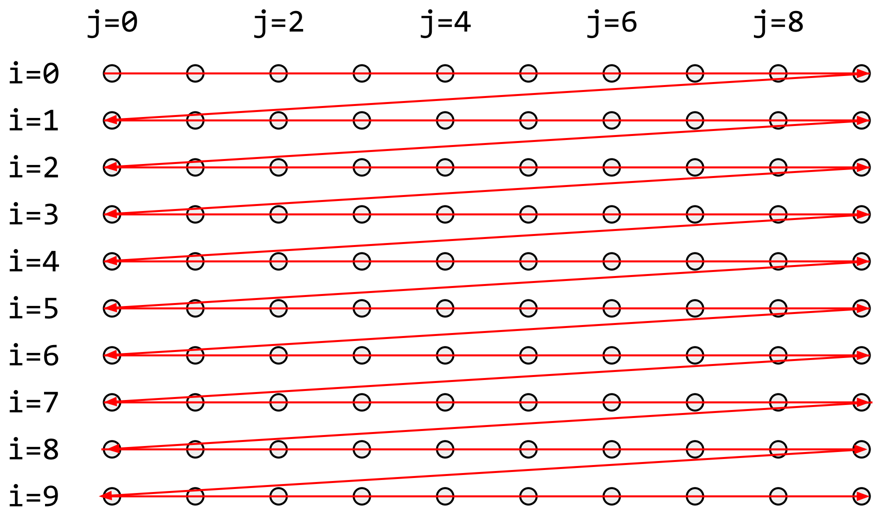
1. Execute *init-statement* (assign control variable)
2. Evaluate *condition*
3. if **false**: stop loop, skip *body-statement*
4. otherwise (**true**)
    a. execute *body-statement*
    b. execute *advance-statement*
    c. go to step 2

```
// print 10 through 15
int i{ 0 };
for (i = 10; i <= 15; ++i) {
   std::cout << i << "\n";
}
```

Output:

```
10
11
12
13
14
15
```

Visualizing Nested Loops



```
int number{0};
for (int i = 0; i < 10; i++) {
  for (int j = 0; j < 10; j++) {
    number = number + 2*j - i;
  }
}
```

Recap: Declaring a std::vector

declaration statement:

std::vector<*data-type***>** *identifier* **{** *element ...* **};**

where

- *data-type* is the type of one element
- *identifier* is variable name
- *element...* are expressions of type *T*
- vector is initialized to store *element ...*
- size is automatically calculated

```
// size 2
std::vector<double> coords{ 1.0, 4.2 };

// size 7
std::vector<int> phone{2,7,8,1,7,1,2};

// size 2
std::vector<bool> truths{true, false};
```

2D Vector (Vector of Vectors)

- Element type *T* of outer vector is another vector type
- Called
  - “Vector of vectors”
  - 2D Vector
  - Matrix

```
std::vector<std::vector<int>> table;
```

Initializing a 2D Vector

statement:

std::vector<std::vector<*T***>>** *ident***(**
*rows***, std::vector<***T***>(***cols*, *value***));**

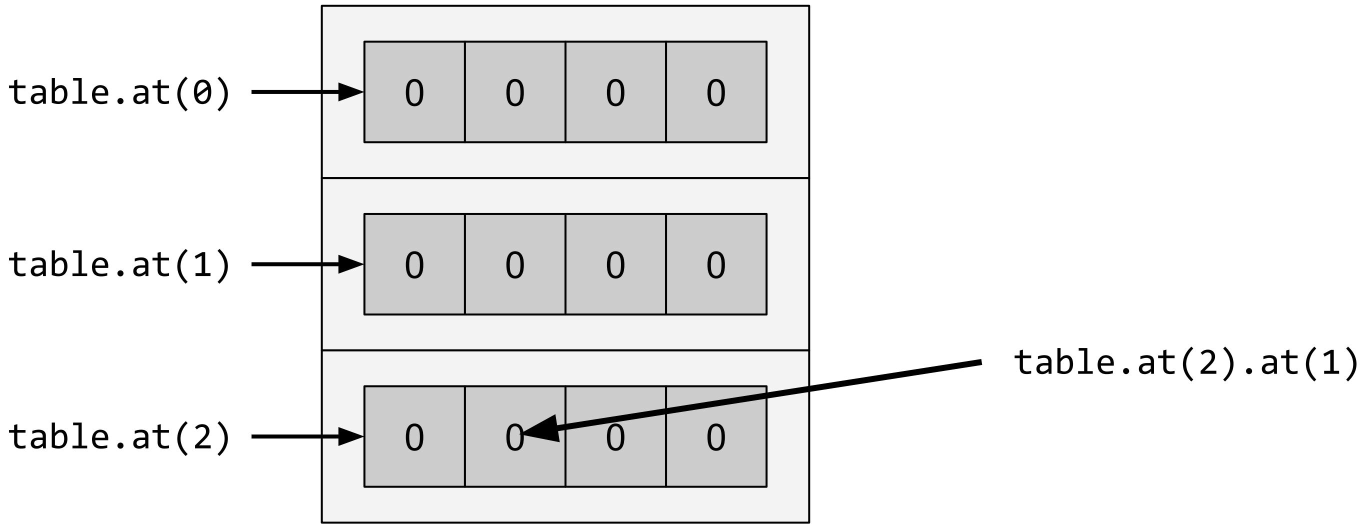
Semantics

- *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));
```

Visualizing a 2D Vector

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



2D Vector Initialization Step By Step

- **Fill constructor**
- *rows* copy of one row
- Sample row is itself created with fill constructor

Vector Fill Constructor

- std::vector::vector variation (3)
- **std::vector(***count***,** *value***)**
- Constructs the vector with *count* copies of *value*
- Ex. construct a vector with 100 copies of 4.0:

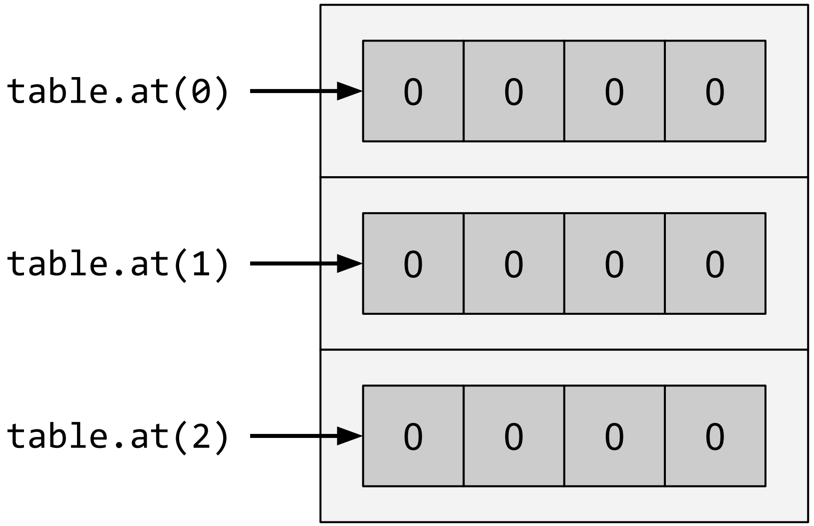
```
std::vector<double> v(100, 4.0);
```

How 2D Vector Initialization Works

std::vector<std::vector<*T*>> *ident*(*rows*, **std::vector<*T*>(***cols*, *value***))**;

- Two nested fill constructors
- Outer: *rows* copies of the inner constructor
- Inner: *cols* copies of *value*

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



Review: Declaring a std::vector

declaration statement:

std::vector<*data-type***>** *identifier* **{** *element ...* **};**

where

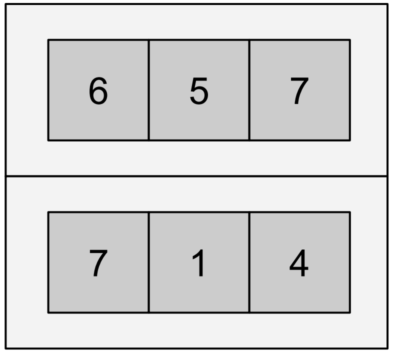
- *data-type* is the type of one element
- *identifier* is variable name
- *element...* are expressions of type *T*
- vector is initialized to store *element ...*
- size is automatically calculated

```
// size 2
std::vector<double> coords{ 1.0, 4.2 };


// size 7
std::vector<int> phone{2,7,8,1,7,1,2};


// size 2
std::vector<bool> truths{true, false};
```

2D Vector Initializer List

- Outer vector *element...* are given between outer { } braces
- Each element of the outer vector is
  - a “row”
  - a vector
  - elements are initialized between { } braces

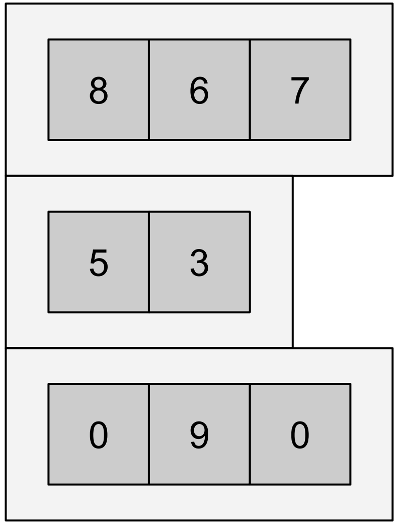
```
std::vector<std::vector<int>> vec{
   {6, 5, 7},
   {7, 1, 4}
 };
```



Pitfall: Jagged Vector

- **Jagged** vector: 2D vector with rows of differing sizes
- Probably unwanted
  - Very rare for business logic to involve a jagged vector
  - Usually a bug
- Omitting an element by mistake in an initializer list causes a jagged vector
- **Best practice:** construct 2D vectors with the fill constructor if possible

```cpp
std::vector<std::vector<int>> jagged{
  {8, 6, 7},
  {5, 3},
  {0, 9, 0}
};
```



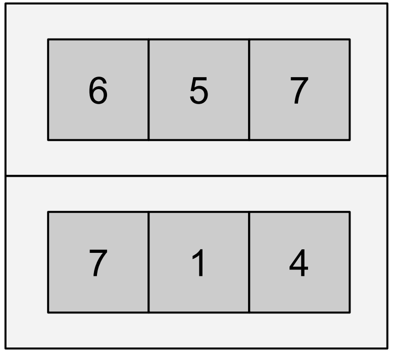
2D Vector Height

- **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:

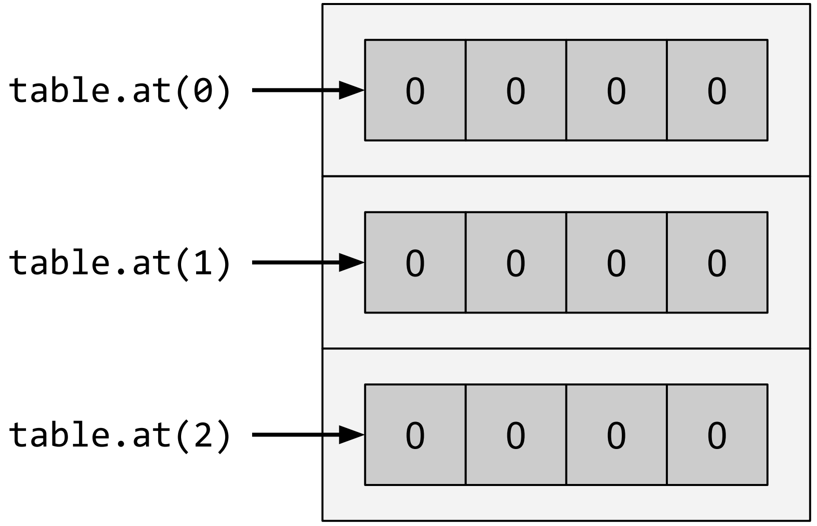
```
height is 2
```



Accessing an Entire Row

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

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



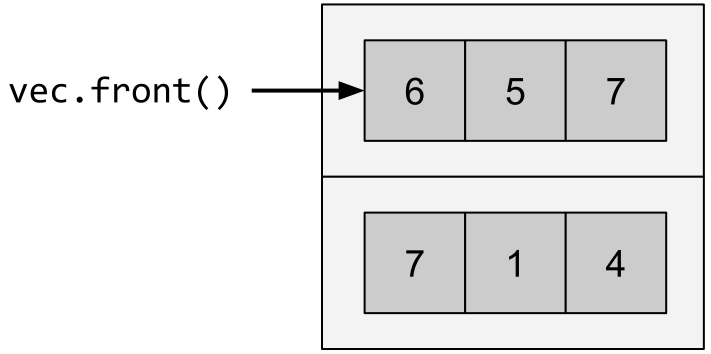
2D Vector Width

- **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
```



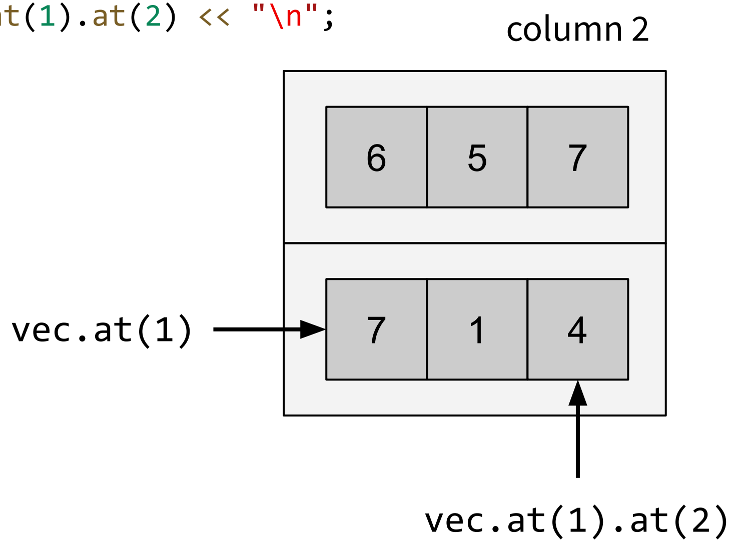
Accessing an Individual Element

- 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 (std::vector<int> row : table) {
   for (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);
  }
}
```

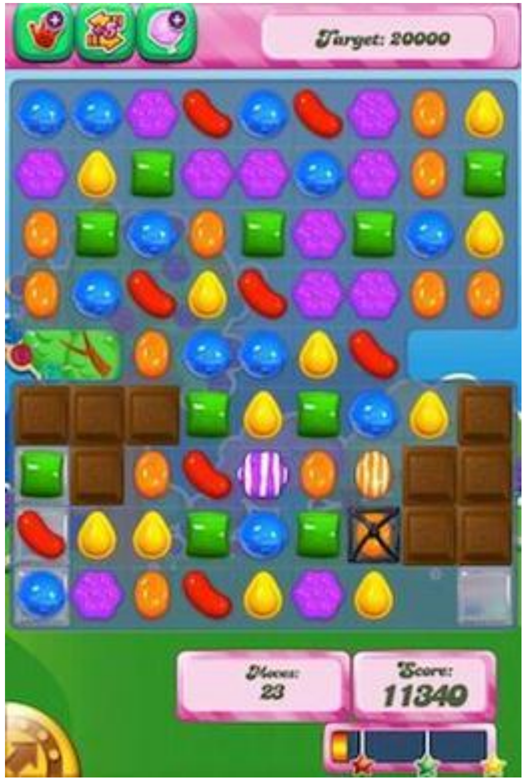
3. Nested Vector Applications

Application: Game Board

- Some games have a **board** corresponding to a 2D vector
    - won’t say “map” because that’s a kind of data structure
- Vector height, width matches game board
- Each element represents one board position
- Could be
    - `int`
    - Constants for each type of board location
    - Later: **class** to represent more game state

Candy Crush Saga

```cpp
const int kBoardWidth{9};
const int kBoardHeight{9};
const int kCellEmpty{0};
const int kCellOffLimits{1};
const int kCellBlue{2};
const int kCellPurple{3};
// ... rest of constants

std::vector<std::vector<int>> board(
    kBoardHeight, std::vector<int>(kBoardWidth, kCellEmpty));
board.at(4).at(0) = kCellOffLimits;
board.at(4).at(1) = kCellOffLimits;
board.at(4).at(7) = kCellOffLimits;
board.at(4).at(8) = kCellOffLimits;
// ... initialize rest of board ...
```



Checkers

```
const int kBoardWidth{8};
const int kBoardHeight{8};
const int kCellEmpty{0};
const int kCellRed{1};
const int kCellBlack{2};

std::vector<std::vector<int>> board(
    kBoardHeight, std::vector<int>(kBoardWidth, kCellEmpty));
// 4 columns of pieces
for (int col = 0; col < kBoardWidth; col += 2) {
  // first three rows of black pieces
  board.at(0).at(col) = kCellBlack;
  board.at(1).at(col + 1) = kCellBlack;
  board.at(2).at(col) = kCellBlack;

  // last three rows of red pieces
  board.at(5).at(col + 1) = kCellRed;
  board.at(6).at(col) = kCellRed;
  board.at(7).at(col + 1) = kCellRed;
}


// play game...
```

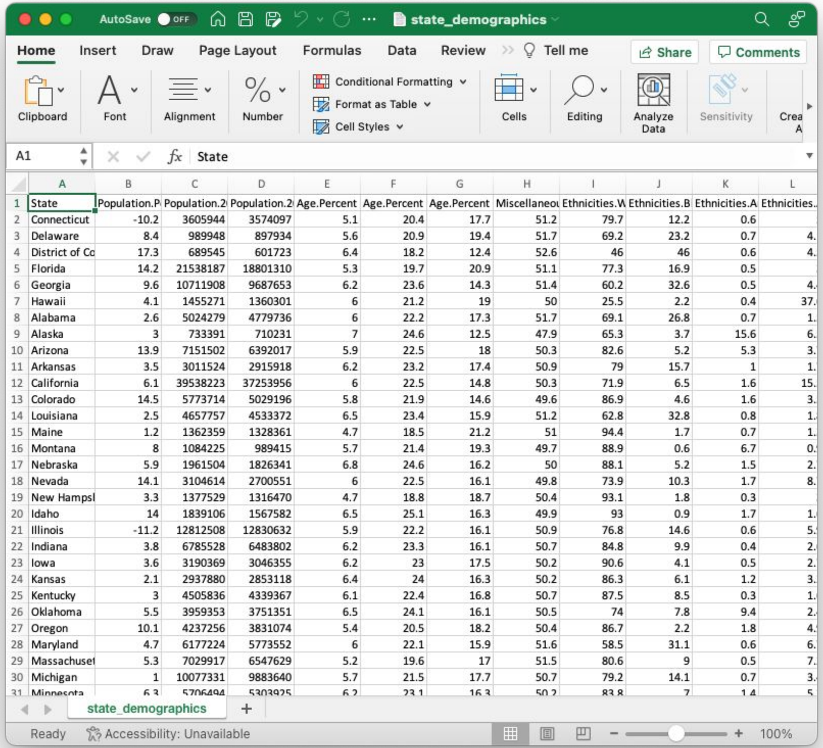
CSV Files

- **CSV** = **C**omma **S**eparated **V**alues
- Common format for data files
- Spreadsheet programs can work with CSV
  - Excel, Google Sheets, Apple Numbers, LibreOffice Calc
- Examples: https://corgis-edu.github.io/corgis/csv/
- **Row** = one line of text
- **Columns** are delimited by commas
- String cells have “” quotes around data
- First row is a **header** with names of fields

CORGIS state_demographics.csv

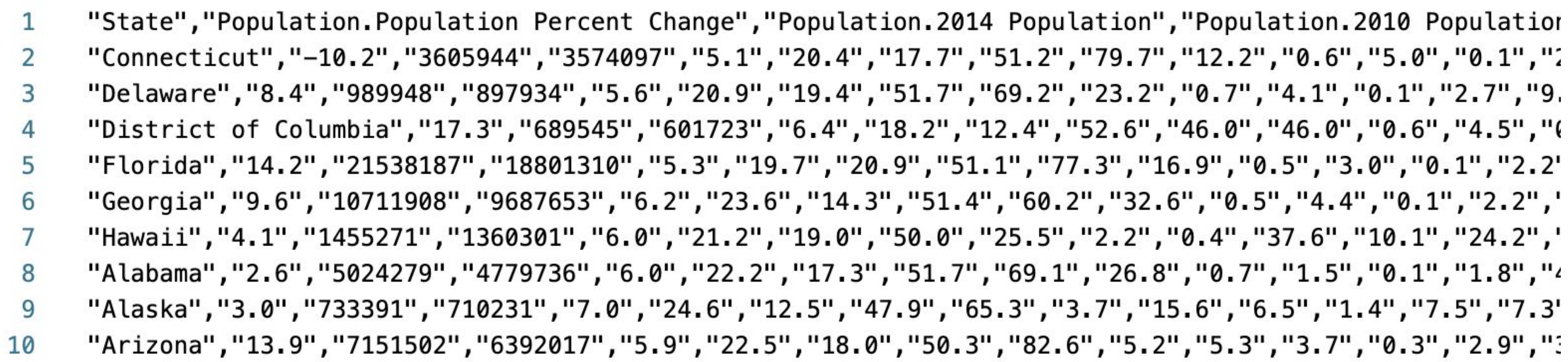
- State Demographics CSV File
- Each row is a US state
  - name
  - population
  - demographics
  - income
  - etc.
- 52 rows
  - 50 states
  - 1 header
  - District of Columbia

state_demographics.csv in Excel



	A	B	C	D	E	F	G	H	I	J	K	L
1	State	Population.P	Population.2	Population.2	Age.Percent	Age.Percent	Age.Percent	Miscellaneou	Ethnicities.W	Ethnicities.B	Ethnicities.A	Ethnicities.
2	Connecticut	-10.2	3605944	3574097	5.1	20.4	17.7	51.2	79.7	12.2	0.6	
3	Delaware	8.4	989948	897934	5.6	20.9	19.4	51.7	69.2	23.2	0.7	4.
4	District of Co	17.3	689545	601723	6.4	18.2	12.4	52.6	46	46	0.6	4.
5	Florida	14.2	21538187	18801310	5.3	19.7	20.9	51.1	77.3	16.9	0.5	
6	Georgia	9.6	10711908	9687653	6.2	23.6	14.3	51.4	60.2	32.6	0.5	4.
7	Hawaii	4.1	1455271	1360301	6	21.2	19	50	25.5	2.2	0.4	37.
8	Alabama	2.6	5024279	4779736	6	22.2	17.3	51.7	69.1	26.8	0.7	1.
9	Alaska	3	733391	710231	7	24.6	12.5	47.9	65.3	3.7	15.6	6.
10	Arizona	13.9	7151502	6392017	5.9	22.5	18	50.3	82.6	5.2	5.3	3.
11	Arkansas	3.5	3011524	2915918	6.2	23.2	17.4	50.9	79	15.7	1	1.
12	California	6.1	39538223	37253956	6	22.5	14.8	50.3	71.9	6.5	1.6	15.
13	Colorado	14.5	5773714	5029196	5.8	21.9	14.6	49.6	86.9	4.6	1.6	3.
14	Louisiana	2.5	4657757	4533372	6.5	23.4	15.9	51.2	62.8	32.8	0.8	1.
15	Maine	1.2	1362359	1328361	4.7	18.5	21.2	51	94.4	1.7	0.7	1.
16	Montana	8	1084225	989415	5.7	21.4	19.3	49.7	88.9	0.6	6.7	0.
17	Nebraska	5.9	1961504	1826341	6.8	24.6	16.2	50	88.1	5.2	1.5	2.
18	Nevada	14.1	3104614	2700551	6	22.5	16.1	49.8	73.9	10.3	1.7	8.
19	New Hampsl	3.3	1377529	1316470	4.7	18.8	18.7	50.4	93.1	1.8	0.3	
20	Idaho	14	1839106	1567582	6.5	25.1	16.3	49.9	93	0.9	1.7	1.
21	Illinois	-11.2	12812508	12830632	5.9	22.2	16.1	50.9	76.8	14.6	0.6	5.
22	Indiana	3.8	6785528	6483802	6.2	23.3	16.1	50.7	84.8	9.9	0.4	2.
23	Iowa	3.6	3190369	3046355	6.2	23	17.5	50.2	90.6	4.1	0.5	2.
24	Kansas	2.1	2937880	2853118	6.4	24	16.3	50.2	86.3	6.1	1.2	3.
25	Kentucky	3	4505836	4339367	6.1	22.4	16.8	50.7	87.5	8.5	0.3	1.
26	Oklahoma	5.5	3959353	3751351	6.5	24.1	16.1	50.5	74	7.8	9.4	2.
27	Oregon	10.1	4237256	3831074	5.4	20.5	18.2	50.4	86.7	2.2	1.8	4.
28	Maryland	4.7	6177224	5773552	6	22.1	15.9	51.6	58.5	31.1	0.6	6.
29	Massachuset	5.3	7029917	6547629	5.2	19.6	17	51.5	80.6	9	0.5	7.
30	Michigan	1	10077331	9883640	5.7	21.5	17.7	50.7	79.2	14.1	0.7	3.
31	Minnesota	6.3	5706494	5303925	6.2	23.1	16.3	50.2	83.8	7	1.4	5.

state_demographics.csv in VS Code



```
"State","Population.Population Percent Change","Population.2014 Population","Population.2010 Populatio
"Connecticut","-10.2","3605944","3574097","5.1","20.4","17.7","51.2","79.7","12.2","0.6","5.0","0.1","
"Delaware","8.4","989948","897934","5.6","20.9","19.4","51.7","69.2","23.2","0.7","4.1","0.1","2.7","9
"District of Columbia","17.3","689545","601723","6.4","18.2","12.4","52.6","46.0","46.0","0.6","4.5","
"Florida","14.2","21538187","18801310","5.3","19.7","20.9","51.1","77.3","16.9","0.5","3.0","0.1","2.2"
"Georgia","9.6","10711908","9687653","6.2","23.6","14.3","51.4","60.2","32.6","0.5","4.4","0.1","2.2",
"Hawaii","4.1","1455271","1360301","6.0","21.2","19.0","50.0","25.5","2.2","0.4","37.6","10.1","24.2",
"Alabama","2.6","5024279","4779736","6.0","22.2","17.3","51.7","69.1","26.8","0.7","1.5","0.1","1.8","
"Alaska","3.0","733391","710231","7.0","24.6","12.5","47.9","65.3","3.7","15.6","6.5","1.4","7.5","7.3"
"Arizona","13.9","7151502","6392017","5.9","22.5","18.0","50.3","82.6","5.2","5.3","3.7","0.3","2.9","
```

Application: Store Entire CSV File

- INPUT: CSV filename
- OUTPUT: 2D vector of strings
- Each CSV cell becomes an element of the 2D vector

Reading CSV Files

- Open file: `std::ifstream`
- Read string cell (including “”): `getline`(*`file`*, *`string-var`*, `‘,’`)
- Skip one comma: *`file`*`.ignore(1, ‘,‘)`

Application: Store Entire CSV File

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

```cpp
 // 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

```cpp
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 (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
...
```