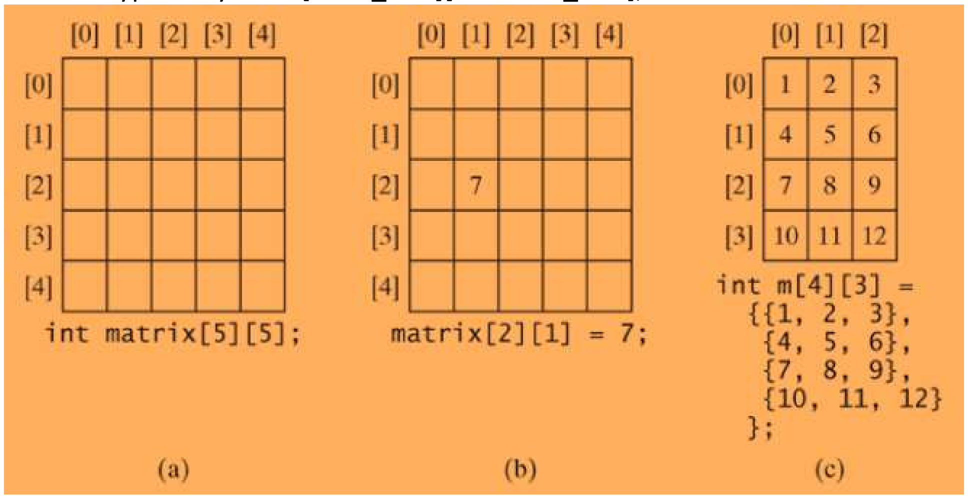
8.2 Declaring Two-Dimensional Arrays

Tuesday, February 21, 2023 1:43 PM

- An element in a 2d array is accessed through a row and column index
- Syntax:
- elementType arrayName[ROW_SIZE][COLUMN_SIZE];
- 

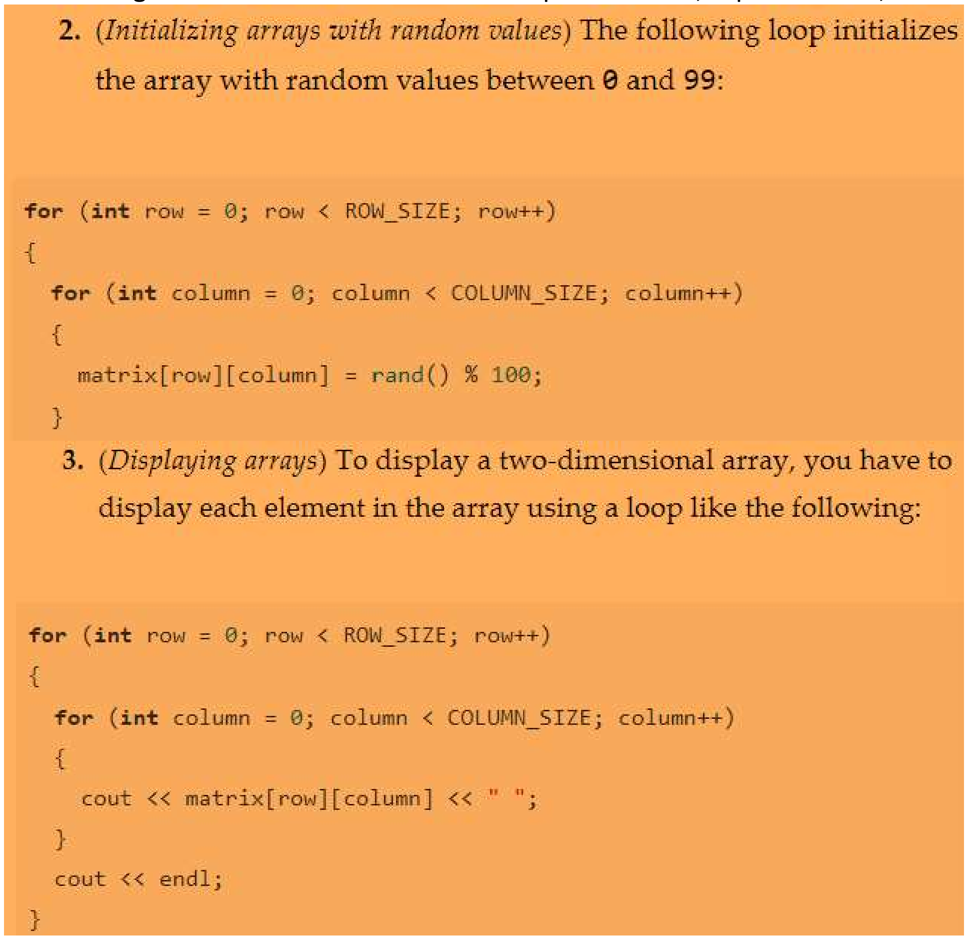
```
int m[4][3] =
  {{1, 2, 3},
   {4, 5, 6},
   {7, 8, 9},
   {10, 11, 12}
  };
```

- To assign specific elemt to specific spot:
  - matrix[2][1] = 7;
  -

8.3 Processing Two-Dimensional Arrays

Tuesday, February 21, 2023 1:55 PM

- Nested for loops used for 2d array processes
-
-
```
const int ROW_SIZE = 10;
const int COLUMN_SIZE = 10;
int matrix[ROW_SIZE][COLUMN_SIZE];
```
-
```
cout << "Enter " << ROW_SIZE << " rows and "
  << COLUMN_SIZE << " columns: " << endl;
for (int i = 0; i < ROW_SIZE; i++)
  for (int j = 0; j < COLUMN_SIZE; j++)
    cin >> matrix[i][j];
```
- Now it assigns whatever user wants to each spot in matrix, top to bottom, but first is left to right
-
-



2. (*Initializing arrays with random values*) The following loop initializes the array with random values between `0` and `99`:

```
for (int row = 0; row < ROW_SIZE; row++)
{
  for (int column = 0; column < COLUMN_SIZE; column++)
  {
    matrix[row][column] = rand() % 100;
  }
```

3. (*Displaying arrays*) To display a two-dimensional array, you have to display each element in the array using a loop like the following:

```
for (int row = 0; row < ROW_SIZE; row++)
{
  for (int column = 0; column < COLUMN_SIZE; column++)
  {
    cout << matrix[row][column] << " ";
  }
  cout << endl;
}
```

4. (*Summing all elements*) Use a variable named `total` to store the sum. Initially `total` is `0`. Add each element in the array to `total` using a loop like this:

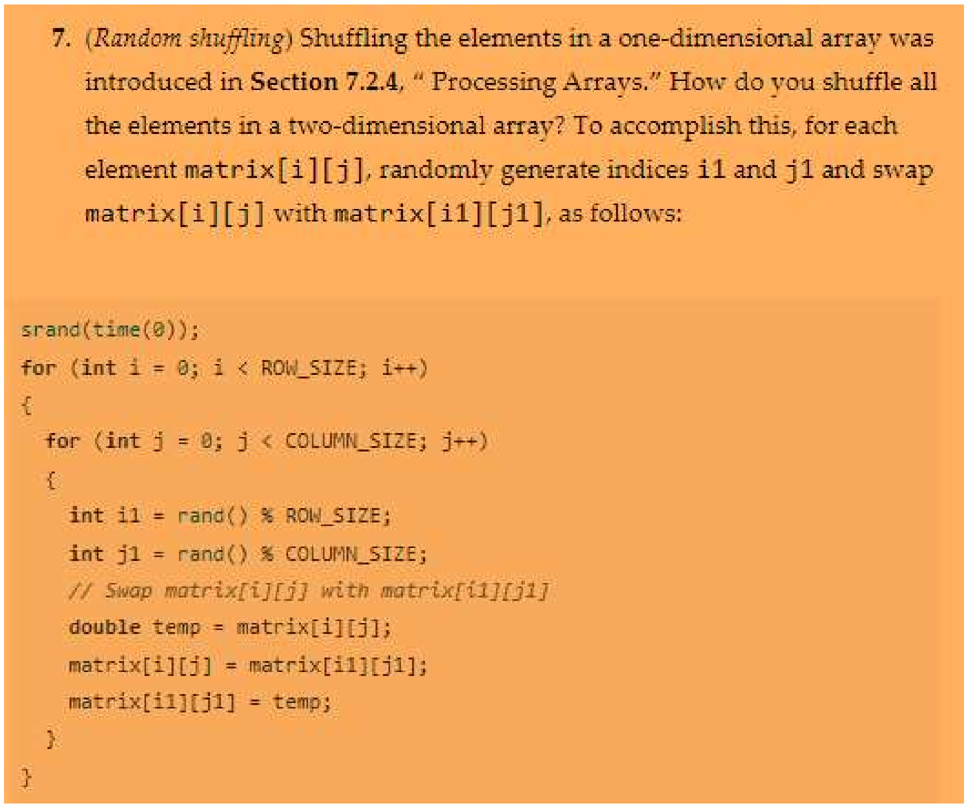
-
  ```
int total = 0;
for (int row = 0; row < ROW_SIZE; row++)
{
  for (int column = 0; column < COLUMN_SIZE; column++)
  {
    total += matrix[row][column];
  }
}
```

5. (*Summing elements by column*) For each column, use a variable named `total` to store its sum. Add each element in the column to `total` using a loop like this:

-
  ```
for (int column = 0; column < COLUMN_SIZE; column++)
{
  int total = 0;
  for (int row = 0; row < ROW_SIZE; row++)
    total += matrix[row][column];
  cout << "Sum for column " << column << " is " << total << endl;
}
```

6. (*Which row has the largest sum?*) Use variables `maxRow` and `indexOfMaxRow` to track the largest sum and index of the row. For each row, compute its sum and update `maxRow` and `indexOfMaxRow` if the new sum is greater.

-
  ```
int maxRow = 0;
int indexOfMaxRow = 0;
// Get sum of the first row in maxRow
for (int column = 0; column < COLUMN_SIZE; column++)
  maxRow += matrix[0][column];
for (int row = 1; row < ROW_SIZE; row++)
{
  int totalOfThisRow = 0;
  for (int column = 0; column < COLUMN_ SIZE; column++)
    totalOfThisRow += matrix[row][column];
  if (totalOfThisRow > maxRow)
  {
    maxRow = totalOfThisRow;
    indexOfMaxRow = row;
  }
}
```



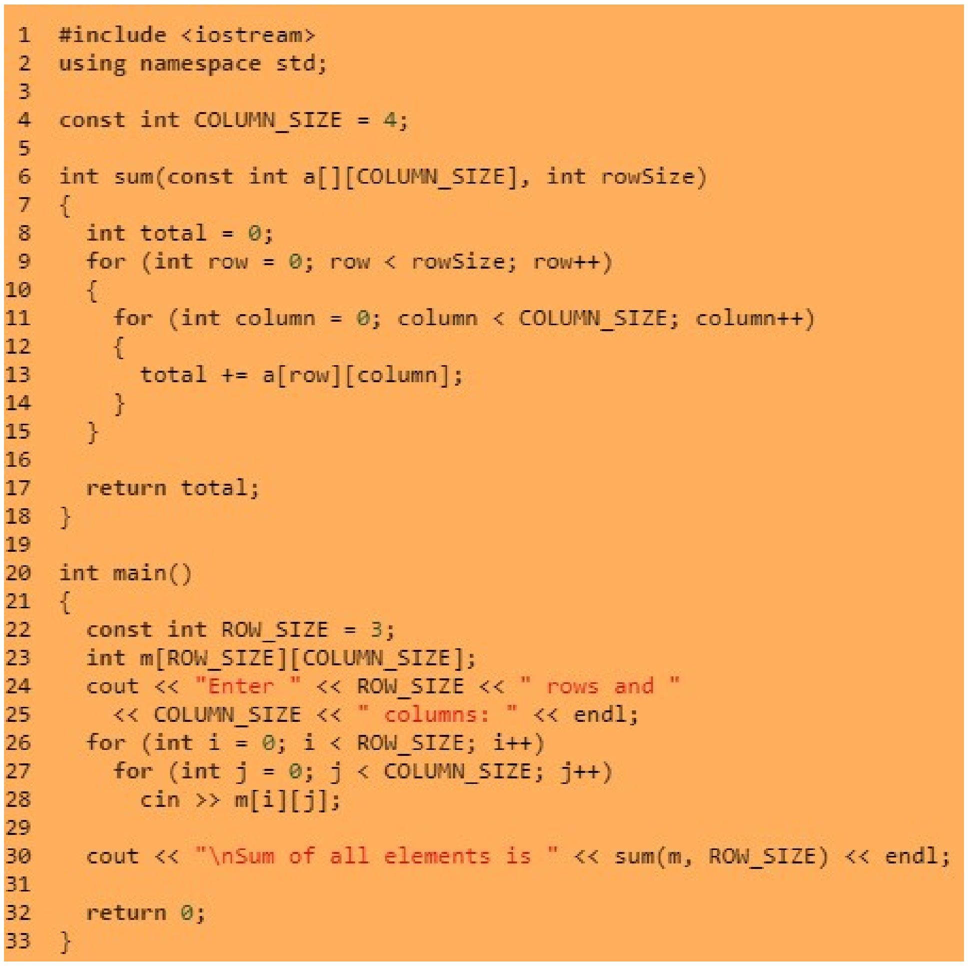
7. (*Random shuffling*) Shuffling the elements in a one-dimensional array was introduced in **Section 7.2.4**, " Processing Arrays." How do you shuffle all the elements in a two-dimensional array? To accomplish this, for each element `matrix[i][j]`, randomly generate indices `i1` and `j1` and swap `matrix[i][j]` with `matrix[i1][j1]`, as follows:

```
srand(time(0));
for (int i = 0; i < ROW_SIZE; i++)
{
  for (int j = 0; j < COLUMN_SIZE; j++)
  {
    int i1 = rand() % ROW_SIZE;
    int j1 = rand() % COLUMN_SIZE;
    // Swap matrix[i][j] with matrix[i1][j1]
    double temp = matrix[i][j];
    matrix[i][j] = matrix[i1][j1];
    matrix[i1][j1] = temp;
  }
}
```

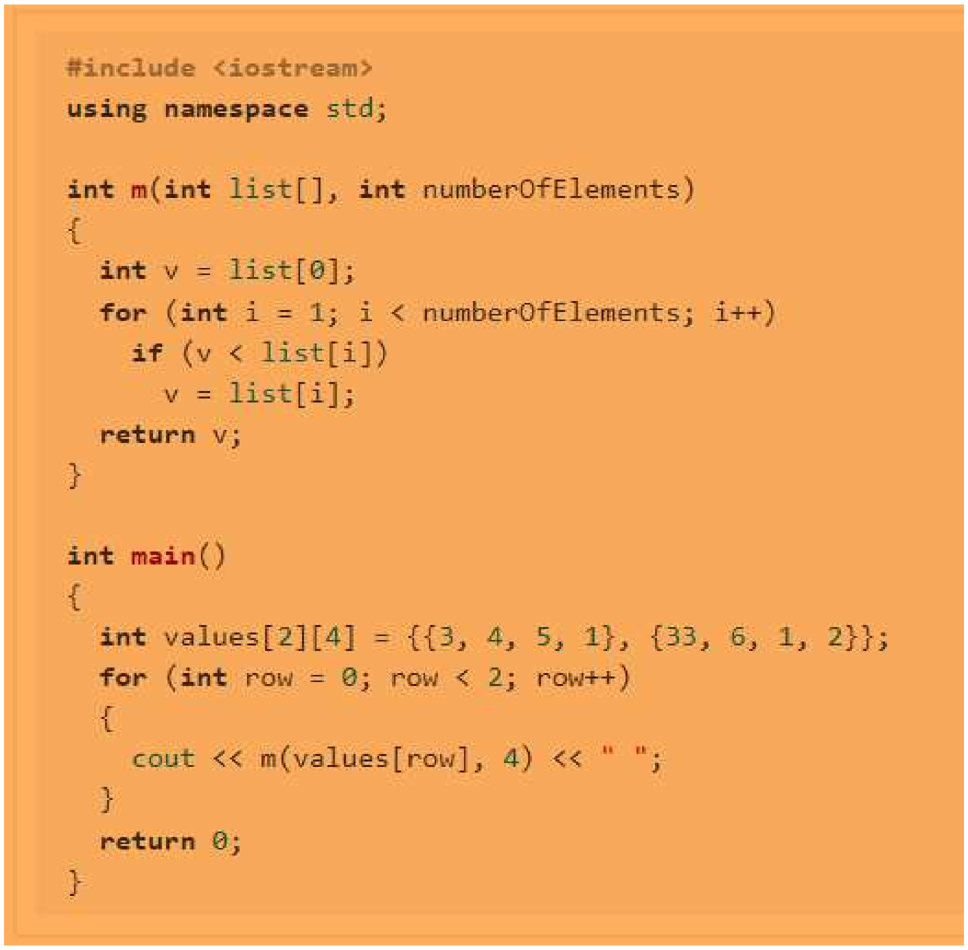
8.4 Passing 2d Array to Functions

Wednesday, February 22, 2023 10:12 AM

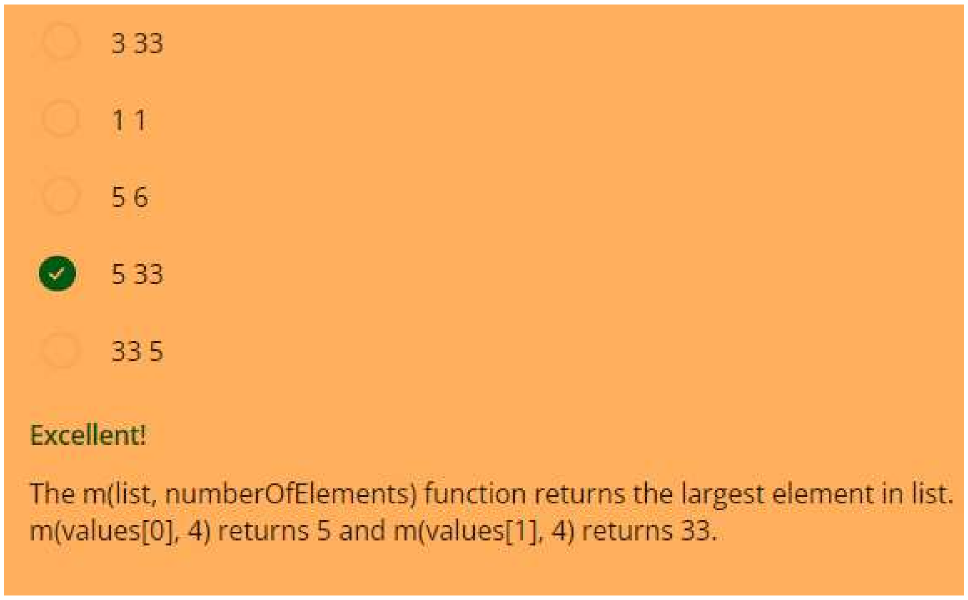
- When passing 2d array to a function, C++ req column size be specified in function parameter type declaration



```cpp
#include <iostream>
using namespace std;

const int COLUMN_SIZE = 4;

int sum(const int a[][COLUMN_SIZE], int rowSize)
{
  int total = 0;
  for (int row = 0; row < rowSize; row++)
  {
    for (int column = 0; column < COLUMN_SIZE; column++)
    {
      total += a[row][column];
    }
  }

  return total;
}

int main()
{
  const int ROW_SIZE = 3;
  int m[ROW_SIZE][COLUMN_SIZE];
  cout << "Enter " << ROW_SIZE << " rows and "
    << COLUMN_SIZE << " columns: " << endl;
  for (int i = 0; i < ROW_SIZE; i++)
    for (int j = 0; j < COLUMN_SIZE; j++)
      cin >> m[i][j];

  cout << "\nSum of all elements is " << sum(m, ROW_SIZE) << endl;

  return 0;
}
```

- 

```cpp
#include <iostream>
using namespace std;

int m(int list[], int numberOfElements)
{
  int v = list[0];
  for (int i = 1; i < numberOfElements; i++)
    if (v < list[i])
      v = list[i];
  return v;
}

int main()
{
  int values[2][4] = {{3, 4, 5, 1}, {33, 6, 1, 2}};
  for (int row = 0; row < 2; row++)
  {
    cout << m(values[row], 4) << " ";
  }
  return 0;
}
```

-
- 

  3 33

  1 1

  5 6

  [✓ 5 33]

  33 5

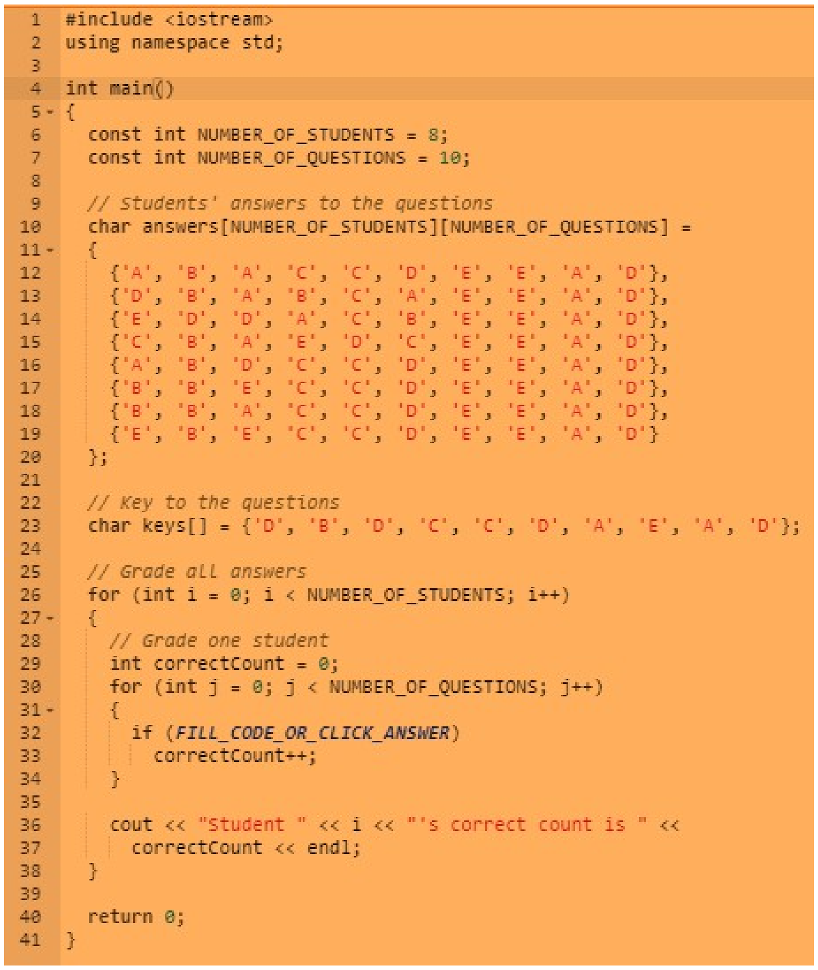
  Excellent!

  The m(list, numberOfElements) function returns the largest element in list. m(values[0], 4) returns 5 and m(values[1], 4) returns 33.

-

8.5 Case Study: Grading a Multiple Choice Test

Friday, February 24, 2023 11:38 AM

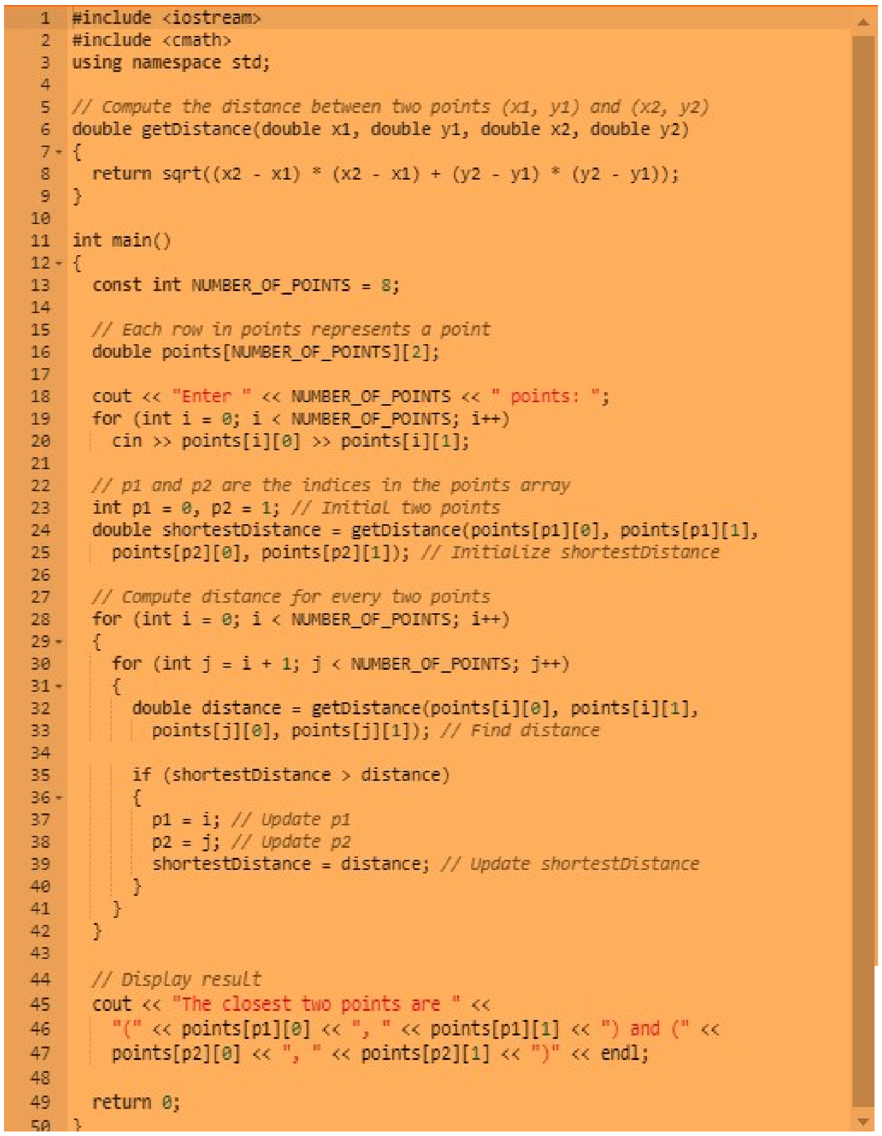
- Prgm that grade multiple choice test
- "8 students and 10 Q's", they stored in 2d array
- Key stored in 1d array
- Pgrm grades test and displays result, compares each student's answers w/ the key, counts numb of correct answers and displays it
- 

```cpp
#include <iostream>
using namespace std;

int main()
{
  const int NUMBER_OF_STUDENTS = 8;
  const int NUMBER_OF_QUESTIONS = 10;

  // Students' answers to the questions
  char answers[NUMBER_OF_STUDENTS][NUMBER_OF_QUESTIONS] =
  {
    {'A', 'B', 'A', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
    {'D', 'B', 'A', 'B', 'C', 'A', 'E', 'E', 'A', 'D'},
    {'E', 'D', 'D', 'A', 'C', 'B', 'E', 'E', 'A', 'D'},
    {'C', 'B', 'A', 'E', 'D', 'C', 'E', 'E', 'A', 'D'},
    {'A', 'B', 'D', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
    {'B', 'B', 'E', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
    {'B', 'B', 'A', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
    {'E', 'B', 'E', 'C', 'C', 'D', 'E', 'E', 'A', 'D'}
  };

  // Key to the questions
  char keys[] = {'D', 'B', 'D', 'C', 'C', 'D', 'A', 'E', 'A', 'D'};

  // Grade all answers
  for (int i = 0; i < NUMBER_OF_STUDENTS; i++)
  {
    // Grade one student
    int correctCount = 0;
    for (int j = 0; j < NUMBER_OF_QUESTIONS; j++)
    {
      if (FILL_CODE_OR_CLICK_ANSWER)
        correctCount++;
    }

    cout << "Student " << i << "'s correct count is " <<
      correctCount << endl;
  }

  return 0;
}
```

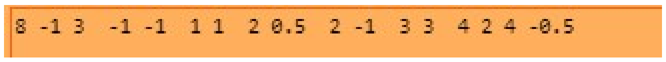
-
- Immediately after student graded, result displayed for the student

8.6 Case Study: Finding a Closest Pair

Friday, February 24, 2023 11:43 AM

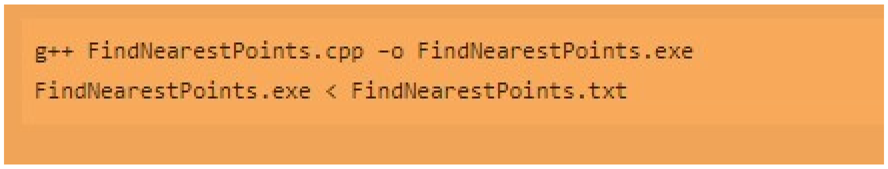
- Section give geometric prob for finding closest pair of points
- Given set of pts, closest pair prob
- 

```cpp
#include <iostream>
#include <cmath>
using namespace std;

// Compute the distance between two points (x1, y1) and (x2, y2)
double getDistance(double x1, double y1, double x2, double y2)
{
  return sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1));
}

int main()
{
  const int NUMBER_OF_POINTS = 8;

  // Each row in points represents a point
  double points[NUMBER_OF_POINTS][2];

  cout << "Enter " << NUMBER_OF_POINTS << " points: ";
  for (int i = 0; i < NUMBER_OF_POINTS; i++)
    cin >> points[i][0] >> points[i][1];

  // p1 and p2 are the indices in the points array
  int p1 = 0, p2 = 1; // Initial two points
  double shortestDistance = getDistance(points[p1][0], points[p1][1],
    points[p2][0], points[p2][1]); // Initialize shortestDistance

  // Compute distance for every two points
  for (int i = 0; i < NUMBER_OF_POINTS; i++)
  {
    for (int j = i + 1; j < NUMBER_OF_POINTS; j++)
    {
      double distance = getDistance(points[i][0], points[i][1],
        points[j][0], points[j][1]); // Find distance

      if (shortestDistance > distance)
      {
        p1 = i; // Update p1
        p2 = j; // Update p2
        shortestDistance = distance; // Update shortestDistance
      }
    }
  }

  // Display result
  cout << "The closest two points are " <<
    "(" << points[p1][0] << ", " << points[p1][1] << ") and (" <<
    points[p2][0] << ", " << points[p2][1] << ")" << endl;

  return 0;
}
```

- 

```
8 -1 3  -1 -1  1 1  2 0.5  2 -1  3 3  4 2 4 -0.5
```

- Pts read from console, stored in 2d array (named points)
- Pgrm uses var shortestDistance to store distance btwn 2 nearest pts, indices of these 2 pts in points array stored in p1 and p2
- Then finds all distances btwn points [i] and [j] for all j>I, when shorter found, saved in var shortestDistance, and p1 & p2 are updated
- Distance btwn 2 pts (x1,y1) and (x2,y2) found w/ length formula
- $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
- If want to, can just enter txt file instead of input all pts, mod to code is:

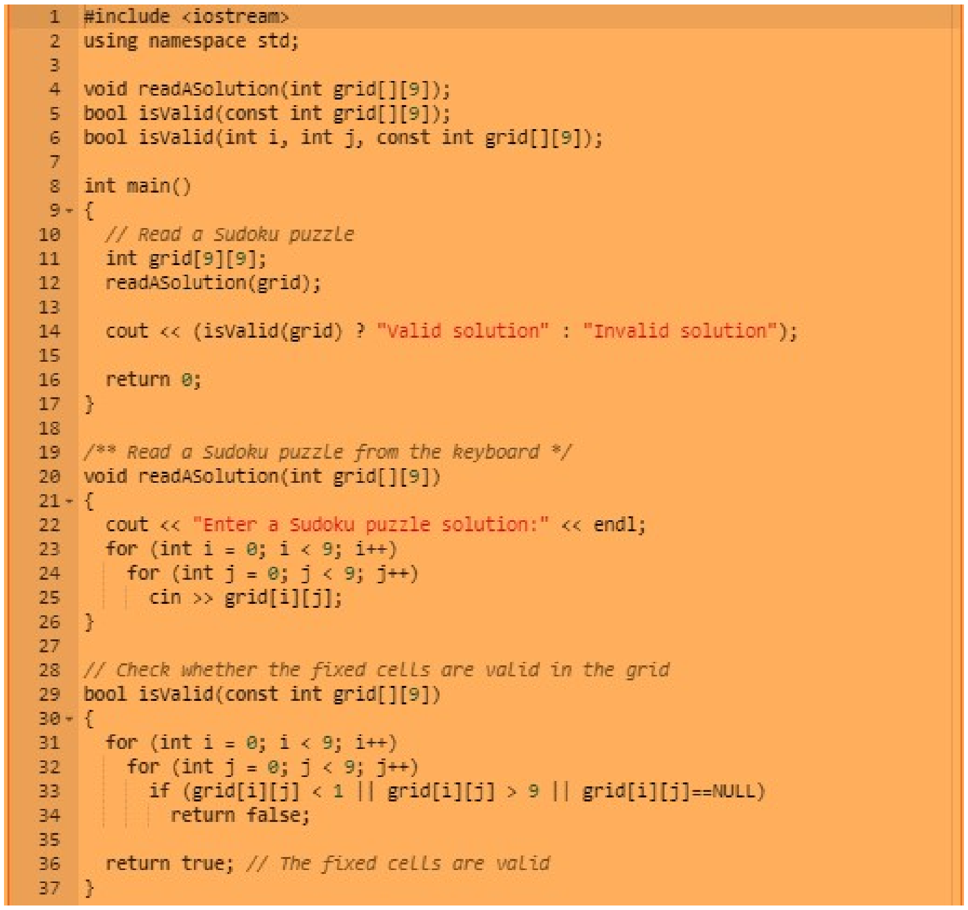
- 

```
g++ FindNearestPoints.cpp -o FindNearestPoints.exe
FindNearestPoints.exe < FindNearestPoints.txt
```

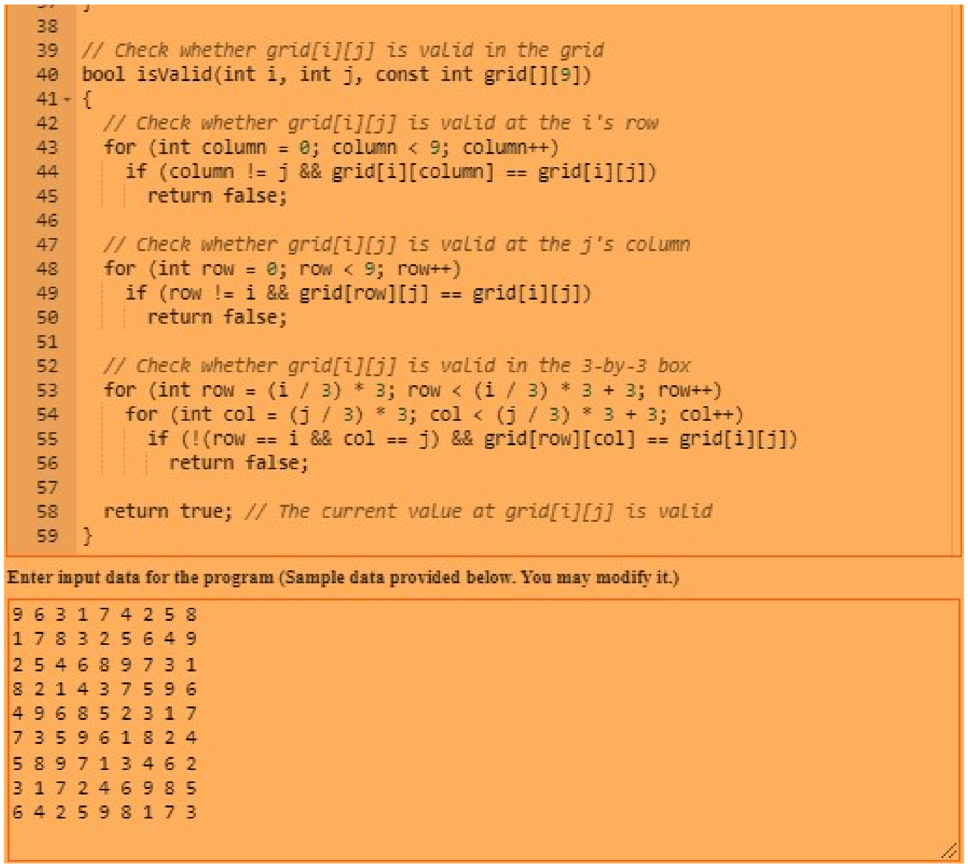
-

8.7 Case Study: Sudoku

Friday, February 24, 2023 12:04 PM

- Check whether given Sudoku solution is correct
- Fill all squares with all the numbers, and all the rows and columns with the numbers, and there can't be duplicates in either
- 2 way to check (9x9 grid, 9 big squares that are 3x3 each)
    - See if every row has numbers 1-9, every column has numbs 1-9, and every big square has numbs from 1-9
    - Check each cell, each cell must be numb 1-9, each cell unique on every row & column & small box
- 

```cpp
#include <iostream>
using namespace std;

void readASolution(int grid[][9]);
bool isValid(const int grid[][9]);
bool isValid(int i, int j, const int grid[][9]);

int main()
{
  // Read a Sudoku puzzle
  int grid[9][9];
  readASolution(grid);

  cout << (isValid(grid) ? "Valid solution" : "Invalid solution");

  return 0;
}

/** Read a Sudoku puzzle from the keyboard */
void readASolution(int grid[][9])
{
  cout << "Enter a Sudoku puzzle solution:" << endl;
  for (int i = 0; i < 9; i++)
    for (int j = 0; j < 9; j++)
      cin >> grid[i][j];
}

// Check whether the fixed cells are valid in the grid
bool isValid(const int grid[][9])
{
  for (int i = 0; i < 9; i++)
    for (int j = 0; j < 9; j++)
      if (grid[i][j] < 1 || grid[i][j] > 9 || grid[i][j]==NULL)
        return false;

  return true; // The fixed cells are valid
}
```

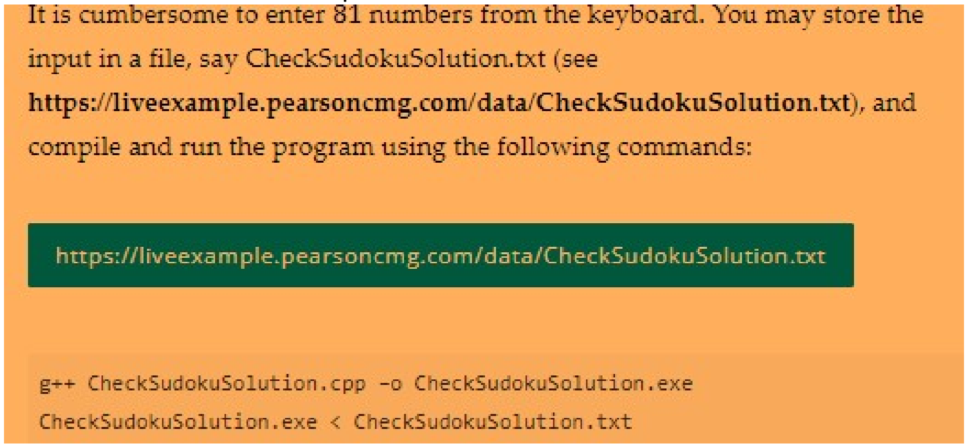
- 

```
}

// Check whether grid[i][j] is valid in the grid
bool isValid(int i, int j, const int grid[][9])
{
  // Check whether grid[i][j] is valid at the i's row
  for (int column = 0; column < 9; column++)
    if (column != j && grid[i][column] == grid[i][j])
      return false;

  // Check whether grid[i][j] is valid at the j's column
  for (int row = 0; row < 9; row++)
    if (row != i && grid[row][j] == grid[i][j])
      return false;

  // Check whether grid[i][j] is valid in the 3-by-3 box
  for (int row = (i / 3) * 3; row < (i / 3) * 3 + 3; row++)
    for (int col = (j / 3) * 3; col < (j / 3) * 3 + 3; col++)
      if (!(row == i && col == j) && grid[row][col] == grid[i][j])
        return false;

  return true; // The current value at grid[i][j] is valid
}
```

Enter input data for the program (Sample data provided below. You may modify it.)

```
9 6 3 1 7 4 2 5 8
1 7 8 3 2 5 6 4 9
2 5 4 6 8 9 7 3 1
8 2 1 4 3 7 5 9 6
4 9 6 8 5 2 3 1 7
7 3 5 9 6 1 8 2 4
5 8 9 7 1 3 4 6 2
3 1 7 2 4 6 9 8 5
6 4 2 5 9 8 1 7 3
```

- IsValid(grid) checks if values in the grid are valid
- Can also store numbers in input txt file

- 

It is cumbersome to enter 81 numbers from the keyboard. You may store the input in a file, say CheckSudokuSolution.txt (see **https://liveexample.pearsoncmg.com/data/CheckSudokuSolution.txt**), and compile and run the program using the following commands:

https://liveexample.pearsoncmg.com/data/CheckSudokuSolution.txt

```
g++ CheckSudokuSolution.cpp -o CheckSudokuSolution.exe
CheckSudokuSolution.exe < CheckSudokuSolution.txt
```

-
-