## Exam 2 - Results

<table>
<thead>
<tr>
<th></th>
<th>MC</th>
<th>Code</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>max</td>
<td>80 (100%)</td>
<td>120 (100%)</td>
<td>196 (98%)</td>
</tr>
<tr>
<td>min</td>
<td>24 (30%)</td>
<td>0 (0%)</td>
<td>24 (12%)</td>
</tr>
<tr>
<td>mean</td>
<td>67.5 (84.5%)</td>
<td>95 (79.3%)</td>
<td>162.8 (81.4%)</td>
</tr>
<tr>
<td>median</td>
<td>68 (85%)</td>
<td>103 (85.8%)</td>
<td>172 (86%)</td>
</tr>
<tr>
<td>stdev</td>
<td>9.4</td>
<td>24</td>
<td>30.5</td>
</tr>
</tbody>
</table>
Exam 2 - Code - frequencies

mean: 95
median: 103
mean: 67.5
median: 68
E2 – Code+MC Frequencies

mean: 162.8  
median: 172
Two-Dimensional Arrays

- 2-D Arrays
- 2-D Array Processing
- Passing 2-D Arrays
- Arrays of Arrays
Arrays

- Implements list or vector data
- Basic array is one-dimensional
  - $x x x x x x x x x x ...$
  - $0 1 2 3 4 5 6 7 8 ...$
Two-Dimensional Arrays

- Implement table data
- Example uses:
  - Matrix
  - Image
  - Game board
  - Spreadsheet
a 1-D array of int

const int NUMSCORES = 50;

int scores [NUMSCORES];

scores [ 0 ]

scores [ 1 ]

scores [ 49 ]

scores
a 2-D array of char

(char ticTacToe[3][3];

(tic-tac-toe)
a 2-D array of char

char tictactoe[3][3];

Place an ‘X’ in the middle

tictactoe[1][1] = ‘X’; // really ttt[row][col]

(rows)

(cols)

(tic-tac-toe)
**a 2-D array of char**

```plaintext
char tictactoe[3][3];
```

Place an ‘X’ in the middle

tictactoe[1][1] = ‘X’;  // really ttt[row][col]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ttt[0][0]</td>
<td>ttt[0][1]</td>
<td>ttt[0][2]</td>
</tr>
<tr>
<td>1</td>
<td>ttt[1][0]</td>
<td>ttt[1][1]</td>
<td>ttt[1][2]</td>
</tr>
<tr>
<td>2</td>
<td>ttt[2][0]</td>
<td>ttt[2][1]</td>
<td>ttt[2][2]</td>
</tr>
</tbody>
</table>

(rows)

(cols)

(tic-tac-toe)
Example: 2-D Array of double

idea:

• Record amount spent each day of the week
• on food, school supplies, & entertainment
<table>
<thead>
<tr>
<th>indices</th>
<th>0 (food)</th>
<th>1 (supplies)</th>
<th>2 (entertain.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (Sun)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 (Mon)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 (Tue)</td>
<td></td>
<td>25.00</td>
<td></td>
</tr>
<tr>
<td>3 (Wed)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 (Thur)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 (Fri)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 (Sat)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example: 2-D Array of double

idea:
• Record amount spent each day of the week
• on food, school supplies, & entertainment

const int NUMDAYS = 7;
const int NUMCATS = 3;
double budget[NUMDAYS][NUMCATS];
int row, column;
### budget

<table>
<thead>
<tr>
<th>indices</th>
<th>0 (food)</th>
<th>1 (supplies)</th>
<th>2 (entertain.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (Sun)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 (Mon)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 (Tue)</td>
<td></td>
<td>25.00</td>
<td></td>
</tr>
<tr>
<td>3 (Wed)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 (Thur)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 (Fri)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 (Sat)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- there are $7 \times 3 = 21$ cells
- `budget[row][column]` accesses 1 element
- `budget[2][0] = 25.00;`

would set the cell shown
Accessing 2-D budget Array

<table>
<thead>
<tr>
<th>indices</th>
<th>0 (food)</th>
<th>1 (supplies)</th>
<th>2 (entertain.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (Sun)</td>
<td>budget[0][0]</td>
<td>budget[0][1]</td>
<td>budget[0][2]</td>
</tr>
<tr>
<td>1 (Mon)</td>
<td>budget[1][0]</td>
<td>budget[1][1]</td>
<td>budget[1][2]</td>
</tr>
<tr>
<td>2 (Tue)</td>
<td>budget[2][0]</td>
<td>budget[2][1]</td>
<td>budget[2][2]</td>
</tr>
<tr>
<td>3 (Wed)</td>
<td>budget[3][0]</td>
<td>budget[3][1]</td>
<td>budget[3][2]</td>
</tr>
<tr>
<td>4 (Thur)</td>
<td>budget[4][0]</td>
<td>budget[4][1]</td>
<td>budget[4][2]</td>
</tr>
<tr>
<td>5 (Fri)</td>
<td>budget[5][0]</td>
<td>budget[5][1]</td>
<td>budget[5][2]</td>
</tr>
<tr>
<td>6 (Sat)</td>
<td>budget[6][0]</td>
<td>budget[6][1]</td>
<td>budget[6][2]</td>
</tr>
</tbody>
</table>

- Indices to each cell, shown in the form row, column
Initialization

• Initialize all values to zero:
  double budget[NUMDAYS][NUMCATS] = { 0 };

• Initialize all values directly:
  double budget[NUMDAYS][NUMCATS] =
  {
   {20, 20, 40 },
   {20, 20, 10 },
   ...
  };

Problem

• Define a C++ type for representing characters of 300 rows and 400 columns.
Accessing one Row in budget

• Hold the row number constant while varying the column number

```c
for (column = 0; column < NUMCATS; column++)
    budget[2][column] = 0.0;
```

• sets all values of row 2 to zero

access order:

```
2,0  2,1  2,2
```
Accessing a column

- Hold the column number constant while varying the row number

```
for (row = 0; row < NUMDAYS; row++)
    budget[row][2] = 0.0;
```

- Sets all values of column 2 to zero

access order:

```
0,2  1,2  2,2  3,2  4,2  5,2  6,2
```
Passing Arrays

- Recall: one-D arrays passed by reference
- Example:

```c
void processArray ( int[ ] );  //Prototype

int  myArray1 [ARR_SIZE];    // Declaration
...
processArray( myArray1 );    //Call
```
Passing 2-D Arrays

```c
void processArray (int arr[][COLS]); //Prototype

int myArray2[ROWS][COLS];          //Declaration
...
processArray( myArray2);           //Call
```

- Parameter specification:
  ```c
  int [ROWS] [COLS];
  int [ROWS] [];
  int [] [COLS]
  int [] [];
  ```
Why 2nd Dimension Size Required

- Passing array communicates location of 1st element: Array[0][0]

- Arrays stored in row order
  - { (0,0), (0,1), (0,2) … (1,0), (1,1) , … } 

- Where is Array[1][0]?
  - Need to know number of columns
Read in budget Amounts Interactively

- call: `readBudget (budget);`

- possible parameter specs:
  - double `budget[NUMDAYS][NUMCATS]`
  - double `budget[ ] [NUMCATS]`
  - double `[] []`
readBudget function

```cpp
void readBudget ( double budget[NUMDAYS][NUMCATS] )
{
    int row, column;

    for(row = 0; row < NUMDAYS; row++)
    for(column = 0; column < NUMCATS; column++) {
        cout << "Enter an amount for day "
             << row << " and "
             << " category "
             << column << " -> " ;
        cin >> budget [ row ] [ column ];
    }
}
```
about readBudget

• budget array is loaded in this order:
  0,0  0,1  0,2  1,0  1,1  1,2  2,0  2,1
  2,2  3,0 ...

• What would happen if we put the column loop on the outside and ran this code?
void readBudget ( double budget[NUMDAYS][NUMCATS] )
{
    int row, column;

    for(row = 0; row < NUMDAYS; row++)
        for(column = 0; column < NUMCATS; column++)
        {
            cout << "Enter an amount for day "
                 << row << " and "
                 << " category "
                 << column << " -> " ;
            cin >> budget [ row ] [ column ];
        }
}
void readBudget ( double budget[NUMDAYS][NUMCATS] )
{
    int row, column;

    for(column = 0; column < NUMCATS; column++)
      for(row = 0; row < NUMDAYS; row++)
      {
        cout << "Enter an amount for day "
        << row << " and "
        << " category "
        << column << " -> " ;
        cin >> budget[ row ][ column ];
      }
}
### Example Table Data

<table>
<thead>
<tr>
<th>indices</th>
<th>0 (food)</th>
<th>1 (supplies)</th>
<th>2 (entertain.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (Sun)</td>
<td>15.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>1 (Mon)</td>
<td>15.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>2 (Tue)</td>
<td>15.00</td>
<td>5.50</td>
<td>0.00</td>
</tr>
<tr>
<td>3 (Wed)</td>
<td>15.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>4 (Thur)</td>
<td>15.00</td>
<td>57.32</td>
<td>0.00</td>
</tr>
<tr>
<td>5 (Fri)</td>
<td>15.00</td>
<td>0.00</td>
<td>15.00</td>
</tr>
<tr>
<td>6 (Sat)</td>
<td>15.00</td>
<td>3.25</td>
<td>40.00</td>
</tr>
</tbody>
</table>

- **Next task**: Determine amount spent on each category for the week
- **Must access entire array COLUMN-WISE**
- **Should obtain**: 105.00  66.07  55.00
sumCategories function

```cpp
void sumCategories(const double budget[NUMDAYS][NUMCATS])
{
    int row, column;
    double colTotal;
    for (column = 0; column < NUMCATS; column++)
    {
        colTotal = 0.0;
        for (row = 0; row < NUMDAYS; row++)
            colTotal = colTotal +
                budget[ row ][ column ];

        cout << "Total for category "
            << column << " is "
            << colTotal << endl;
    }
}
```
Row-wise and Column-wise

- sumCategories visits array cells in column-wise order:
  0,0 1,0 2,0 3,0 4,0 5,0 6,0 0,1
  1,1 ...

- Exercise: write a function printBudget that prints all elements in the table
Problem

• Given:
• int arr[3][5];

• find the smallest value within arr
Alternative: Array of Arrays

// declare a type representing one row
typedef double Categories [NUMCATS];

// declare a type for the entire 2D array
typedef Categories BudgetTable2 [NUMDAYS];

// declare a variable which is a 2D array
BudgetTable2 budget2;
BudgetTable2

- `budget2` has same basic structure as `budget` declared earlier
- Except: now have type for the 1-D row vector
- Therefore:
  - can pass one row of `budget2`
  - `(as budget2 [ i ] ) using`
  - `Categories` as the parameter type
  - you cannot pass one row of `budget`!