Lecture 15

Arrays III and Enumerated Types
Multidimensional Arrays & enums
Multidimensional Arrays

- So far we have worked with arrays with one dimension.
- Single dimensional arrays can logically be thought of as vectors.
- We can also declare and use multidimensional arrays
- Multidimensional arrays can be logically though of as 2D tables, 3D tables (cubes), ...
- A multidimensional array in memory is just an array of arrays.
- Are usefule when you need to store a table. e.g. tic-tac-toe board, matrices, etc.
Multidimensional Arrays - Declaration

• Declare them as:

\[
\text{type name}[\text{size of d1}][\text{size of d2}][\text{size of d3}]...
\]

char ticTacTao[3][3]

• If we are initializing during declaration, we can omit the size of the first dimension: (only the first)

char ticTacToe[ ][3] = ....
int my3dArray[ ][15][200] = ....

![Diagram of a 3x3 grid representing tic tac toe array indices. ticTacToe[0][0], ticTacToe[2][1], ticTacToe[0][2] are labeled.](image-url)
Multidimensional Arrays - Initialization

char ticTacTao[3][3] = { { ‘ ‘, ‘ ‘, ‘ ‘}, {‘ ‘, ‘ ‘, ‘ ‘}, {‘ ‘, ‘ ‘, ‘ ‘} };
int myTable[2][3] = { {1, 3, 4}, {2, 5, 7} };  
int myTable[][3] = { {1, 3, 4}, {2, 5, 7} };  

- Even when initializing during the declaration the 2\textsuperscript{nd}, 3\textsuperscript{rd}, and higher dimension sizes should be given!
  
  invalid: int myTable[][] = { {1, 3, 4}, {2, 5, 7} };  

- This is true, when using multidimensional arrays as function parameters: only the first dimension size can be eliminated from the header.
  
  int myFunction(int myThreeDimArray[][2][3]);
Traversing Over Multidimensional Arrays

• We can use nested loops to traverse over these arrays
• Each loop will iterate over one dimension of the array.

```cpp
int table[2][3] = {{ 3, 5, 7}, {2, 4, 6}};
for (i = 0; i < 2; i++)
{
    for (j = 0; j < 3; j++)
    {
        sum += table[i][j];
    }
}
```
Example – Matrix Summation

- Assuming we show each matrix as a 2D array, write a program that performs addition for two 2*3 matrices.

\[
\begin{bmatrix}
a_1 & a_2 & a_3 \\
a_4 & a_5 & a_6 \\
\end{bmatrix} + 
\begin{bmatrix}
b_1 & b_2 & b_3 \\
b_4 & b_5 & b_6 \\
\end{bmatrix} = 
\begin{bmatrix}
a_1 + b_1 & a_2 + b_2 & a_3 + b_3 \\
a_4 + b_4 & a_5 + b_5 & a_6 + b_6 \\
\end{bmatrix}
\]
# Example – Matrix Summation

```c
#define NUM_COLUMNS 3

int addMatrices(double a[][NUM_COLUMNS], int aRows,
                 double b[][NUM_COLUMNS], int bRows,
                 int result[][NUM_COLUMNS])
{
    int success = 1, i, j;
    if ( aRows != bRows )
    {
        printf("cannot add two matrices!");
        success = 0;
    }
    else
    {
        for ( i = 0; i < aRows; i++ )
            for ( j = 0; j < NUM_COLUMNS; j++ )
                results[i][j] = a[i][j] + b[i][j];
    }
    return success;
}
```

Writing all dimension sizes except optionally the first one is mandatory
Enumerated Types (enum)

- Sometimes we want to define our own data types rather than using int, double, etc.
- Enumerated types are data types we can create
- (there are other ways to define new data types too)

- We are working with week days.
- One way is to represent each day as an integer
  - Monday = 1, Tuesday = 2 …
- More convenient way is to define a new data type “weekday_t”
**Enumerated Types**

```c
typedef enum
{
    monday, tuesday, wednesday, thursday, friday
} weekday_t;
```

- Now `weekday_t` is a data type and you can use it to declare variables:
  ```c
  weekday_t today = monday;
  (int day = 0);
  ```

- Name of the type we are creating

- Valid values for this data type
Enumerated Types

typedef enum
{
    monday, tuesday, wednesday, thursday, friday
} weekday_t;

• A new data type called weekday_t is defined.
• Valid data values for a variable of type weekday_t are monday, tuesday, wednesday, thursday, and friday
• *Behind the scene monday gets int value 0, tuesday gets 3, wednesday 4, and so on.
Enumerated Types

• We can say:
  weekday_t today = monday;
  weekday_t tomorrow = today + 1;

• We can iterate over an enum using loops:
  weekday_t day;
  for (day = monday; day <= friday; day++)
  {
    ...
  }

• We can also convert from enums to int or vice versa.
  int dayNumber = (int) today;
  weekday_t today = (weekday_t) 3;

• We can use enums as array indices:
  customersCount[monday] = 10;

• We can use them in switch-case operations
Common/Useful enums

• typedef enum

{
    false, true
} Bool;

• typedef enum

{
    jan = 1, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec
} Month;

*By writing jan = 1 in Month enum, the integer association to month identifiers start from 1 rather than 0.
Printing enums

- Enums can be used in switch statements
  - (useful for printing them!)
  ```c
  void printWeekday(weekday_t day)
  {
      switch (day)
      {
          case monday:
              printf("Monday");
              ...
      }
  }
  ```
- Note: C does not provide specific functions/placeholders for writing/reading enum data.
Enum - notes

• Valid values for enums are called “enumeration constants”.
  • monday, tuesday, … in our example are enumerations constants

• Enumeration constants must be identifiers (names), they cannot be constant numbers or “string literals”.

• One identifier cannot be used in more than one enum definition.
Example – Super Market Sales

• A super market keeps records of its amount of daily sales in a week for each of the categories:
  ▪ Cleaning products,
  ▪ Drinks,
  ▪ Patio and garden,
  ▪ Books
  ▪ Pharmacy

(assume there are no row/column headers in the records file. Here we include headers for illustration)

<table>
<thead>
<tr>
<th></th>
<th>mon</th>
<th>tue</th>
<th>wed</th>
<th>thu</th>
<th>fri</th>
<th>sat</th>
<th>sun</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cleaning</td>
<td>230</td>
<td>190</td>
<td>110</td>
<td>42</td>
<td>50</td>
<td>390</td>
<td>540</td>
</tr>
<tr>
<td>Drinks</td>
<td>75</td>
<td>236</td>
<td>559</td>
<td>690</td>
<td>980</td>
<td>430</td>
<td>98</td>
</tr>
<tr>
<td>Garden</td>
<td>37</td>
<td>49</td>
<td>198</td>
<td>290</td>
<td>360</td>
<td>455</td>
<td>61</td>
</tr>
<tr>
<td>Books</td>
<td>660</td>
<td>490</td>
<td>51</td>
<td>230</td>
<td>250</td>
<td>110</td>
<td>230</td>
</tr>
<tr>
<td>Pharmacy</td>
<td>539</td>
<td>426</td>
<td>620</td>
<td>740</td>
<td>18</td>
<td>56</td>
<td>85</td>
</tr>
</tbody>
</table>

• Write a program that prints the highest selling category for each day of the week.
Example – Super Market Sales Solution: enums

```c
#define CATEGORIES_COUNT 5
#define DAYS_COUNT 7

typedef enum {
    mon, tue, wed, thu, fri, sat, sun
} days_t;

typedef enum {
    cleaning, drinks, garden, books, pharmacy
} category_t;
```
Example – Super Market Sales Solution: Reading the records

void readRecords(double sales[DAYS_COUNT][CATEGORIES_COUNT])
{
    category_t category = cleaning;
    day_t day = mon;
    FILE* infile = fopen("supermarket.txt", "r");
    if ( infile != NULL )
    {
        for ( category = cleaning; category <= pharmacy; category++)
        {
            for (day = mon; day <= sun; day ++)
            {
                fscanf(infile, "%lf", &sales[day][category]);
            }
        }
    }
    else
    {
        printf("file cannot be opened!\n");
    }
}
Example – Super Market Sales Solution: Finding the daily max sales

```c
void findMaxSales(double sales[DAYS_COUNT][CATEGORIES_COUNT])
{
    category_t category = cleaning, dailyMaxCategory = cleaning;
    day_t day = mon;
    double dailyMaxAmount = 0.0;
    for (day = mon; day <= sun; day++)
    {
        dailyMaxAmount = 0;
        for (category = cleaning; category <= pharmacy; category++)
        {
            if (sales[day][category] > dailyMaxAmount)
            {
                dailyMaxCategory = category;
                dailyMaxAmount = sales[day][category];
            }
        }
        printf("for day %d, highest selling category was %d, with sale amount of %f\n",
            (int) day, (int) dailyMaxCategory, dailyMaxAmount);
    }
}
```
References


• Andy O’Fallon’s lecture notes for CptS121 (http://eecs.wsu.edu/~aofallon/cpts121)