Introduction to C

Tutorial 7: Arrays
Last Tutorial…

• Loops
Agenda

• Arrays
• Merging arrays
• Two dimensional arrays
Arrays
Arrays vs. Variables

**Variable inside an array**
- Define an array of variables
  ```
  int y[5];
  ```
- Reading
  ```
  total = y[i] * 10;
  ```
- Writing
  ```
  y[i] = 50 * z;
  ```

**Single variable**
- Define single variable
  ```
  int x;
  ```
- Reading
  ```
  total = x * 10;
  ```
- Writing
  ```
  x = 50 * z;
  ```

Every element in the array points to a single variable!
In C, the elements of an array are stored in consecutive blocks in memory.

For example:

```c
int a[10] = {5, 8, 10, 0, 0, 0, 0, 7, 0, 0};
```
Memory snapshot – Array

• If we give `sizeof()` a name of an array, we’ll get the size of the array in bytes.
• If we use an index bigger than N-1 or less than 0, then we will be outside of the array bounds. This will cause the program to crash or behave in an unexpected way.

The responsibility of using a valid index is the Programmer’s!
When do You Need an Array?

1. Calculating the median of heights
2. Calculate the standard deviation of heights
3. Calculating Temperature histogram
4. Printing numbers less than or more than an average
5. Searching through a particular strand of DNA
6. Combining vectors
What are the details of an array?

What is the **type**, location of **first element**, **second element**, and **number of elements** in each array? Assume the arrays are allocated at address 1000.

```plaintext
int array[10];
int total_studs = 10;
double array[total_studs];
#define NUM_STUDENTS 17
float array[NUM_STUDENTS];
double array[100.0];
long array[];
```
Example: Static initialization of an array

What is each array initialized to?

```plaintext
int array[3] = {1, 2, 3};
float array[3] = {1, 2, 3};
int array[3] = {100.0, 2.3, 3.8};
int array[3] = {100};
int array[3] = {0};
int array[] = {10, 20};
int array[3] = {};
```
Example: Initializing an array using a loop.

What is every array initialized to?
Assume N is defined and array’s length is N.

```c
for (i = 0; i < N; i++)
    array[i] = i;

array[0] = 0;
for (i = 1; i < N; i++)
    array[i] = array[i-1]+1;

array[0] = 1;
for (i = 1; i < N; i++)
    array[i] = array[i-1]*i;

for (i = 0; i < N; i += 2)
    array[i] = array[i+1] = i;
```
Example: Exceeding array boundaries

What is wrong about the following loops? Why?

```c
for (i = N; i > 0; i++)
    array[i] = i;
for (i = N-1; i > 0; i++)
    array[i] = i;
for (i = N-1; i > 0; i--)
    array[i] = i;
for (i = N; i > -1; i--)
    array[i-1] = 10*i;
```
Example: Merging arrays
Example: Merging arrays

- Write a program that gets two sets of 6 numbers in increasing order.
- The program must merge the two arrays and print the new merged array, such that it is still in increasing order.
- Example

  Input
  
  1 3 5 7 9 11  
  2 4 6 8 10 12 

  Output

  1 2 3 4 5 6 7 8 9 10 11 12
Merging arrays: Defining the problem

• We enter the sets of numbers in two arrays `a[]` and `b[]`. Their sizes will be `sizeA` and `sizeB`, respectively.

• We’ll fill `c[]` (of size `sizeA+sizeB`) such that it will be sorted, and will contain all of the elements of `a` and `b`.

• We will print `c[]`. 
Union of the arrays?

- Typically, copying one array right after the other will not solve the problem.

```
<table>
<thead>
<tr>
<th>5</th>
<th>7</th>
<th>9</th>
<th>12</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>
```

```
| 5 | 7 | 9 | 12 | 24 | 3 | 6 | 6 | 7 | 8 |
```
Merging arrays: the algorithm

• We will have an index variable for each array – ia, ib, and ic. (We start at 0)
• Compare a[ia] to b[ib]
  – Copy the smaller one to c[ic] and increment ic.
  – If we took the element from a[], then increment ia. Otherwise, increment ib.
• When we get to the end of a[] or b[], we will copy the rest of the other array to c[]
Merging arrays: Example

16 19 25

14 22 37

14 16 19 22 25 37
Why does it look familiar?

• The problem becomes simpler when considering two stacks of cards, where each stack is sorted.
• The stacks have their numbers facing upward.
• Each time, we take the smaller of the two available cards from the pile and put it in a 3rd pile.
• If one of the stacks becomes empty, then we can move the remaining stack into the 3rd pile.
Merging arrays: input

```c
#include <stdio.h>
#define sizeA 6
#define sizeB 6
#define sizeC (sizeA+sizeB)

int main()
{
    int a[sizeA] = {0}, b[sizeB] = {0}, c[sizeC] = {0};
    int ia, ib, ic;

    /* Input of the First series */
    printf("Enter the first series (%d numbers):
", sizeA);
    for (ia = 0; ia < sizeA; ia++)
        scanf("%d", &a[ia]);

    /* Input of the Second series */
    printf("Enter the second series (%d numbers):
", sizeB);
    for (ib = 0; ib < sizeB; ib++)
        scanf("%d", &b[ib]);
```
Merging arrays: implementation with loops

```c
/* Fill the ordered and merged series */
for(ia = ib = ic = 0; (ia < sizeA)&&(ib < sizeB); ic++) {
    if(a[ia] < b[ib]) { /* The element in “b” is greater */
        c[ic] = a[ia];
        ia++;
    }
    else { /* The element in “a” is greater or equal */
        c[ic] = b[ib];
        ib++;
    }
}
for(;ia < sizeA; ia++, ic++)
    c[ic] = a[ia];
for(;ib < sizeB; ib++, ic++)
    c[ic] = b[ib];
```

Copying the remainder of the other array.
Notice: in practice only one of these copies will actually happen.
Merging arrays: printing the result

```c
/* Print the merged series */
printf("The resulting merged series:\n");
for (ic = 0; ic < sizeC; ic++) {
    printf("%d", c[ic]);
    if (ic+1 < sizeC)
        printf(" ");
}
return 0;
```

What does this condition do?
Two dimensional arrays
The arrays that we’ve looked at so far are one dimensional.
Given an index (a whole number, 0 through size of array), there is a cell that fits that index.
It’s possible to represent a vector with an array.
But... how is it possible to represent a matrix using an array?

$$A = \begin{bmatrix}
3 & 12 & 19 \\
8 & 1 & 5 \\
78 & 36 & 65 \\
0 & 4 & 9 \\
7 & 2 & 6
\end{bmatrix}$$
Comparing a Two Dimensional array with a variable

Variable within 2D array

Defining a 2D array

```c
int y[5][3];
```

- Reading

```c
total = y[i][j] * 10;
```

- Writing

```c
y[i][j] = 50 * z;
```

Single variable

- Defining variable

```c
int x;
```

- Reading

```c
total = x * 10;
```

- Writing

```c
x = 50 * z;
```

Every cell/block in a two dimensional array is an individual variable!
Memory snapshot – 2D Array

• In C, elements of the 2D array appear in contiguous memory locations

Example •

```c
int a[5][3] = {{ 3, 12, 19},
               { 8,  1,  5},
               { 78, 36, 65},
               {  0,  4,  9},
               {  7,  2,  6}};
```

Number of rows = height of matrix
Number of columns = width of matrix
What are the properties of the two dimensional array?

What are the sizes and valid index ranges for the rows and columns in each one?

```c
int matrix[10][20];
float matrix[N][M];
long matrix[N][N];
long matrix[N][];
double matrix[][][];
```

M and N are set with a `#define`
### Example: Static initialization of 2D array

What is each array initialized to?

<table>
<thead>
<tr>
<th>Array Initialization</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. int matrix[3][2] = {{1, 2}, {4, 5}, {6, 7}};</td>
<td>2D array with the given values</td>
</tr>
<tr>
<td>2. int matrix[3][2] = {{10, 9, 8}, {5, 4, 3}};</td>
<td>2D array with the given values</td>
</tr>
<tr>
<td>3. int matrix[3][2] = {{10, 20}};</td>
<td>2D array with the given values</td>
</tr>
<tr>
<td>4. int matrix[3][2] = {{1},{2},{3}};</td>
<td>2D array with the given values</td>
</tr>
<tr>
<td>5. int matrix[3][2] = {{1}};</td>
<td>2D array with the given values</td>
</tr>
<tr>
<td>6. int matrix[3][2] = {1};</td>
<td>2D array with the given values</td>
</tr>
<tr>
<td>7. int matrix[3][2] = {1,2,3,4,5};</td>
<td>2D array with the given values</td>
</tr>
<tr>
<td>8. int matrix[3][2] = {};</td>
<td>2D array with the given values</td>
</tr>
</tbody>
</table>
Example: Almost magic

- A Partial Magic square is a 2D array where the sums of the rows are equal.
  - (There is no requirement for columns or diagonals)
- **Example:** the following:

```
5  6  3
3  8  3
7 -1  8
```

This is a Partial Magic square, since each row has a sum of 14.

- **Exercise:** Write a code snippet that assumes the existence of a 2D array of size N by N, and prints “Partial Magic” if the array is a Partial Magic square. Else, it prints “No Magic”.
- **Assume** that N is defined in a `#define`
Exercise - solution

Variables defined at the beginning of the program

```c
int curr_row_sum, i, j, first_row_sum = 0;
int A[N][N];
```

Code that checks whether A[] is Partial Magic

```c
for (j=0; j<N; j++) {
    first_row_sum += A[0][j];
}
for (i=1; i<N; i++) {
    curr_row_sum = 0;
    for (j=0; j<N; j++) {
        curr_row_sum += A[i][j];
    }
    if (curr_row_sum != first_row_sum) {
        printf("No-Magic");
        return 0;
    }
}
printf("Partial-Magic");
```

We check row by row, that the sum of each row matches the first row.

If sum is different, square is not partial magic. No point to check the rest.

If we got here, then each row sum was equal.

Calculate the sum of the first row

Check sum of the j-th row
Example: Rotating a picture
Example: Rotating a picture

• Write a program that receives an image of size NxN and prints the output image with a rotation angle of 90 degrees clockwise.

• Assume that N is defined in with #define.
Example of a rotation

• For example, input for N=6:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
<td>9</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>9</td>
<td>12</td>
<td>15</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td>30</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>21</td>
<td>28</td>
<td>35</td>
<td>42</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

• The output will be:

<table>
<thead>
<tr>
<th>6</th>
<th>7</th>
<th>5</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>14</td>
<td>10</td>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>21</td>
<td>15</td>
<td>9</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>28</td>
<td>20</td>
<td>12</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>35</td>
<td>25</td>
<td>15</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>42</td>
<td>30</td>
<td>18</td>
<td>12</td>
<td>6</td>
</tr>
</tbody>
</table>
The foundation of the solution

• Assume that the picture was input in the 2D array `in[]`
• Let’s think about what to put in the 2D output array `out[]`.
• To fill a matrix, we must assign values for each i and j.

```java
for (i=0; i<N; i++) {
    for (j=0; j<N; j++) {
        out[i][j] = in[?][?];
    }
}
```

• Now all we have to do is find the proper indexes of `in[]`. 
How do we find the indexes of `in`?

To find the indexes, we should first look at a simple example:

```
0 1 2
i

in[0][0] | in[0][1] | in[0][2]
---------|---------|---------
in[1][0] | in[1][1] | in[1][2]
in[2][0] | in[2][1] | in[2][2]
```

- The first index is the same in each column. That is to say, it does not depend on `i`.
- Each row index starts at 2, and decreases in size as `j` grows.
- In a 4x4 matrix, it would start at 3, in 5x5 it would start at 4 → general formula: the first index on the column `j` is \((N-1-j)\)
- The second index is simply `i`
Solution – inputting a picture

```c
#include <stdio.h>
#define N ...

int main()
{
    int in[N][N], int out[N][N];
    int i, j;

    for (i=0; i<N; i++) {
        for (j=0; j<N; j++) {
            scanf("%d", &in[i][j]);
        }
    }
}
```
```c
for (i=0; i<N; i++) {
    for (j=0; j<N; j++) {
        out[i][j]=in[N-1-j][i];
    }
}

for (i=0; i<N; i++) {
    for (j=0; j<N; j++) {
        printf("%d ", out[i][j]);
    }
    printf("\n");
}

return 0;
}
```