Introduction to C

Tutorial 7: Arrays
Last tutorial...

- loops
Agenda

• Arrays
• Merging arrays
• Two dimensional arrays
Arrays
### Comparing an array to a variable

<table>
<thead>
<tr>
<th>Variable inside an array</th>
<th>Single variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define an array of variables</td>
<td>Define single variable</td>
</tr>
<tr>
<td><code>int y[5];</code></td>
<td><code>int x;</code></td>
</tr>
<tr>
<td>Reading</td>
<td>Reading</td>
</tr>
<tr>
<td><code>total = y[i] * 10;</code></td>
<td><code>total = x * 10;</code></td>
</tr>
<tr>
<td>Writing</td>
<td>Writing</td>
</tr>
<tr>
<td><code>y[i] = 50 * z;</code></td>
<td><code>x = 50 * z;</code></td>
</tr>
</tbody>
</table>

Every element in the array points to a single variable!
Memory snapshot - array

- 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, 0, 0, 0};
```
Memory snapshot – array (cont’d)

• 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 result in the program crashing or unexpected behavior.

The responsibility of using a valid index is on the programmer!
When do you need an array?

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

What is the type, location of first element, second element, and length of the array “array” in each?

```c
int array[10];
int total_studs = 10;
double array[total_studs];
float array[NUM_STUDENTS];
double array[100.0];
long array[];
```

NUM_STUDENTS is defined with a #define
Example: Static initialization of an array

What is each array initialized to?

```c
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?

```
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 takes in two sets of 6 numbers in increasing order
- The program must merge the two arrays and print it, 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.
Merging arrays: the algorithm

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

16 19 25 48 50 67

14 22 37 45 56 61

14 16 19 22 25 37 45 48 50 56 61 67
Why is this similar?

• 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 lesser 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]);
```
/* 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

/* 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;
}
Two dimensional arrays
Arrays, Vectors, and Matrices

• 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
  – Accordingly, in some languages they call them “vectors” instead of arrays
• 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
  ```
  int y[5][3];
  ```
- Reading
  ```
  total = y[i][j] * 10;
  ```
- Writing
  ```
  y[i][j] = 50 * z;
  ```
  Row index
  Column index

Single variable
- Defining variable
  ```
  int x;
  ```
- Reading
  ```
  total = x * 10;
  ```
- Writing
  ```
  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}};
```

<table>
<thead>
<tr>
<th>Row 0</th>
<th>Row 1</th>
<th>Row 2</th>
<th>Row 3</th>
<th>Row 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>12</td>
<td>19</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>78</td>
<td>36</td>
<td>65</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>7</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>
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?

```c
int matrix[3][2] = {{1, 2}, {4, 5}, {6, 7}};
int matrix[3][2] = {{10, 9, 8}, {5, 4, 3}};
int matrix[3][2] = {{10, 20}};
int matrix[3][2] = {{1}, {2}, {3}};
int matrix[3][2] = {{0}};
int matrix[3][2] = {0};
int matrix[3][2] = {1, 2, 3, 4, 5};
int matrix[3][2] = {};
```
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
Variables defined at the beginning of the program:

```c
int curr_row_sum, i, j;
int 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");
```

- **Calculate the sum of the first row**: We check row by row, that the sum of each row matches the first row.
- **Check sum of the \(i^{th}\) 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**: If we got here, then each row sum was equal.
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 \texttt{in[]}.
• Let’s think about what to put in the 2D output array \texttt{out[]}.
• To fill a matrix, we must assign values for each \texttt{i} and \texttt{j}.

```cpp
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 \texttt{in[]}.
How do we find the indexes of \textbf{in}? 

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

- 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 \( \Rightarrow \) general formula is \((N-j-1)\)
#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]);
        }
    }
}
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;