Introduction to C

Tutorial 7: Arrays
Agenda

- Arrays
- Merging arrays
- Two dimensional arrays
Arrays

• An array is a collection of variables of the same type

• Lets say we want to write a program that does something with 1000 integers. We could write:

  ```
  int x0, x1, x2, x3, x4, ..., x999;
  ```

• Or we could define 1000 integers all at once:

  ```
  int x[1000];
  ```

• The integers are named: x[0], x[1], ... x[999]
Arrays vs. Variables

Variable inside an array

- Define an array of variables
  ```c
  int a[5];
  ```
- Read
  ```c
  total = a[i] * 10;
  ```
- Write
  ```c
  a[i] = 10 * z;
  ```

Single variable

- Define a single variable
  ```c
  int x;
  ```
- Read
  ```c
  total = x * 10;
  ```
- Write
  ```c
  x = 10 * z;
  ```

Every element in the array points to a single variable!
// define an array of size 2
int arr[2];

// what we just did is like declaring 2
// variables at once and those variables are
// called arr[0] and arr[1]

// now give those variables values
arr[0] = 5;
arr[1] = 10;

int sum = arr[0] + arr[1];
// sum is now 15
Indexes

- In code like `array[5]` the number 5 is called an index – an index specifies which place in the array we want to access.

- Given an array of size N, never use an index bigger than N-1 or less than 0!
  - If you do, the program will crash or behave in an unexpected way.
Array Initialization

You can assign initial values like this:

```c
int a[3];
a[0] = 5;
a[1] = 10;
a[2] = 15;
```

Or you can assign values when declaring the array:

```c
int a[3] = {5, 10, 15};
```

But you can **NOT** do this:

```c
int a[3];
a = {5, 10, 15};
```
More Array Initialization

There is a trick to quickly initialize all elements to zero:

```c
int a[3] = {0};
```

That is the same as:

```c
int a[3];
a[0] = 0;
a[1] = 0;
a[2] = 0;
```
Printing an array

```c
int arr[3] = {5, 10, 15};

The first value in the array is `arr[0]` and not `arr[1]`! The last value is `arr[2]` and not `arr[3].`

```c
printf("arr[0] is %d\n", arr[0]); // 5
printf("arr[1] is %d\n", arr[1]); // 10
printf("arr[2] is %d\n", arr[2]); // 15
```

NEVER do this because `arr[3]` doesn’t exist!

```c
int a[3] = {5, 10, 15};
printf("arr[3] is %d\n", arr[3]);
```

If you have an array `arr` of size `N` the last value is at `arr[N-1]`!"
We can use a variable as an index!

```c
int arr[3] = {5, 10, 15};
int i = 1;
printf("arr[i] is %d\n", arr[i]); // 10
```

This is useful! Here we print the whole array:

```c
for (int i=0; i<3; i++)
{
    printf("%d\n", arr[i]);
}
```
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, 7, 0, 0};
```
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;
```

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

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

```c
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?

```plaintext
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 >= 0; i--)
    array[i-1] = i;
```
Example: Printing numbers in reverse

• Write a program that takes as input 11 numbers and then prints them in the reverse order

• Example

10 20 30 40 50 60 70 80 90 100 110

110 100 90 80 70 60 50 40 30 20 10

Input

Output
Example: Printing numbers in reverse

```c
int numbers[11];
int i;
for (i=0; i<11; i++)
{
    scanf("%d", &numbers[i]);
}

for (i=10; i>=0; i--)
{
    printf("%d ", numbers[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 \( a[\cdot] \) and \( b[\cdot] \). Their sizes will be sizeA and sizeB, respectively.

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

• We will print \( c[\cdot] \).
Union of the arrays?

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

| 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

/* 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++;
    }
}

What is missing?
Merging arrays: implementation with loops

/* 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
- But... how is it possible to represent a matrix using an array?

$$A = \begin{bmatrix} 3 & 12 & 19 \\ 8 & 1 & 5 \\ 78 & 36 & 65 \\ 0 & 4 & 9 \\ 7 & 2 & 6 \end{bmatrix}$$
Arrays vs. Variables

**Variable inside a 2d array**
- Define a 2d array of variables
  ```
  int a[4][5];
  ```
- Read
  ```
  total = a[i][j] * 10;
  ```
- Write
  ```
  a[i][j] = 10 * z;
  ```

**Single variable**
- Define a single variable
  ```
  int x;
  ```
- Read
  ```
  total = x * 10;
  ```
- Write
  ```
  x = 10 * z;
  ```

Every cell in the 2d array is a single variable!
Array Initialization

You can assign initial values like this:

```c
int a[3][2]; // 3 rows 2 columns
a[0][0] = 1; // top left corner
a[0][1] = 2; // top right corner
a[1][0] = 3;
a[1][1] = 4;
a[2][0] = 5;
a[2][1] = 6; // bottom right corner
```

You can assign values when declaring the array:

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

M and N are set with a #define
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-

<table>
<thead>
<tr>
<th>5</th>
<th>6</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>-1</td>
<td>8</td>
</tr>
</tbody>
</table>

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

We check the sum of each row matches the first row.

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

• The output will be:

<table>
<thead>
<tr>
<th></th>
<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>
<td></td>
</tr>
<tr>
<td>6</td>
<td>21</td>
<td>15</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>28</td>
<td>20</td>
<td>12</td>
<td>8</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>35</td>
<td>25</td>
<td>15</td>
<td>10</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>42</td>
<td>30</td>
<td>18</td>
<td>12</td>
<td>6</td>
<td></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 \texttt{in}?

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

\begin{array}{ccc}
\text{in[0][0]} & \text{in[0][1]} & \text{in[0][2]} \\
\text{in[1][0]} & \text{in[1][1]} & \text{in[1][2]} \\
\text{in[2][0]} & \text{in[2][1]} & \text{in[2][2]} \\
\end{array}

\begin{array}{ccc}
\text{0} & \text{in[2][0]} & \text{in[1][0]} & \text{in[0][0]} \\
\text{1} & \text{in[2][1]} & \text{in[1][1]} & \text{in[0][1]} \\
\text{2} & \text{in[2][2]} & \text{in[1][2]} & \text{in[0][2]} \\
\end{array}

- The first index is the same in each column. That is to say, it does not depend on \texttt{i}.
- Each row index starts at 2, and decreases in size as \texttt{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 \texttt{j} is \((N-1-j)\)
- The second index is simply \texttt{i}
#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;