CSE 142
Computer Programming I
Arrays

Overview
Concepts this lecture
Data structures
Arrays
Subscripts (indices)

Chapter 8
8.1 Declaration and Referencing
8.2 Subscripts
8.3 Loop through arrays
8.4 & 8.5 Arrays arguments and parameters
8.6 Example

Rainfall Data Revisited
General task: Read daily rainfall amounts and print some interesting information about them.

Input data: Zero or more numbers giving daily rainfall followed by a negative number (sentinel).

Example input data:
0.2 0.0 0.0 1.5 0.3 0.0 0.1 -1.0
Empty input sequence:
-1.0

Rainfall Analysis
Possible things to report:
- How many days worth of data are there?
- How much rain fell on the day with the most rain?
- On how many days was there no rainfall?
- What was the average rainfall over the period?
- On how many days was the rainfall above average?
- What was the median rainfall?
Thought question: Can we do all of these with the Read Until Sentinel pattern, i.e., in one pass?

Rainfall Analysis (cont)
For some tasks (median, number of days above average), we need to have all the data before we can do the analysis.

Where do we store the data?
- Lots of variables (rain1, rain2, rain3, rain4, ...)?
- Awkward
- Doesn’t scale

Need something better
Data Structures

Functions give us a way to organize programs. Data structures are needed to organize data, especially:
- large amounts of data
- variable amounts of data
- sets of data where the individual pieces are related to one another

In this course, we will structure data using
- arrays
- structs
- combinations of arrays and structs

Arrays

**Definition:** A named, ordered collection of variables of identical type

- **Name** the collection (`rain`);
- **number** the elements (0 to 6)

Example: rainfall for one week

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.0</td>
</tr>
<tr>
<td>1</td>
<td>0.2</td>
</tr>
<tr>
<td>double</td>
<td></td>
</tr>
<tr>
<td>rain[7];</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>1.4</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
</tr>
</tbody>
</table>

Accessing Variables

Rainfall for one week

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.0</td>
</tr>
<tr>
<td>1</td>
<td>0.2</td>
</tr>
<tr>
<td>double</td>
<td></td>
</tr>
<tr>
<td>rain[7];</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>1.4</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
</tr>
</tbody>
</table>

Variable access:
- `rain[0]` is 1.0
- `rain[6]` is 0.0
- `2.0 * rain[4]` is 2.8

Array Declaration Syntax

- `type name[size];`

- `double rain[7];`

- *type* must be an `int` constant

Array Terminology

- `double rain[7];`
  - `rain` is of type `array of double` with size 7.
  - `rain[0], rain[1], ...`, `rain[6]` are the elements of the array `rain`. Each is a variable of type `double`.
  - 0, 1, ..., 6 are the indices of the array. Also called subscripts.

  - The *bounds* are the lowest and highest values of the subscripts (here: 0 and 6).

Rainfall Analysis (cont.)

Strategy for processing data if we need all of it before we can process it:
1. Read data and store it in an array
2. Analyze data stored in the array

Key detail: In addition to the array, we need to keep track of how much of the array currently contains valid data.
Keeping Track of Elements In-Use

Since an array has to be declared a fixed size, you often declare it bigger than you think you'll really need:

```
#define MAXRAINDAYS 400
int rain[MAXRAINDAYS];
```

How do you know which elements in the array actually hold data, and which are unused extras?

1. Keep the valid entries together at the front
2. Record number of valid entries in a separate variable

---

Print # Days Above Average

Algorithm:
- Read data into an array
- Compute average rainfall (from array)
- Keeping track of total # of days
- Count # days above average (from array)
- Print result

---

Declarations

```
/* Maximum # of days of input data */
#define MAXRAINDAYS 400
int main(void) {       /* rainfall data is stored in */
    /* rain[0..numRainDays-1] */
    double rain[MAXRAINDAYS];
    int numRainDays ;
    double rainfall;      /* current input value */
    double rainTotal;     /* sum of input rainfall values */
    double rainAverage;   /* average rainfall */
    /* # days with above average rainfall */
    int numAbove;
    int k;
```

---

Read Data Into Array

```
/* read and store rainfall data */
printf("Please enter rainfall data.\n");
numRainDays = 0;
scanf("%lf", &rainfall);
while (rainfall >= 0.0) {
    rain[numRainDays] = rainfall;
    numRainDays++;
    scanf("%lf", &rainfall);
}
```

---

Calculate Average

```
double rain[MAXRAINDAYS]; /* rainfall data*/
int numRainDays;
/* # of data values */
double rainTotal;     /* sum of input values*/
double rainAverage;   /* average rainfall*/
int k;
/* calculate average rainfall */
rainTotal = 0;
for (k = 0; k < numRainDays; k++) {
    rainTotal = rainTotal + rain[k];
}
rainAverage = rainTotal / numRainDays;
```

We should make a test to avoid a divide by zero
Calculate and Print Answer

double rain[MAXRAINDAYS]; /* rainfall data*/
int numRainDays; /* # of data values */
double rainAverage; /* average rainfall */
int numAbove; /* # of days above average */
int k;
/* count # of days with rainfall above average */
numAbove = 0;
for (k = 0; k < numRainDays; k++) {
    if (rain[k] > rainAverage)
        numAbove++;}
/* Print the result */
printf("%d days above the average of %.3f.
", numAbove, rainAverage);

Index Rule

Rule: An array index must evaluate to an int between 0 and n-1, where n is the number of elements in the array. No exceptions!

Example:
    rain[i+3+k] /* OK as long as 0 \leq i+3+k \leq 6 */

The index may be very simple
rain[i]
or incredibly complex
rain[(int) (3.1 * fabs(sin (2.0*PI*sqrt(29.067))))]

Do you think C will help you with this?
Do you think it should?

C Array Bounds are Not Checked

#define DAYS_IN_WEEK 7
double rain[DAYS_IN_WEEK];
int index;
index = 900;
... 
rain[index] = 3.5; /* Is index out of range?? */

You need to be sure that the subscript value is in range. Peculiar and unpleasant things can (and probably will) happen if it isn't.

Technicalities

An array is a collection of variables
Each element can be used wherever a simple variable of that type is allowed.
- Assignment, expressions, input/output
- An entire array can't be treated as a single variable in C
  - Can't assign or compare arrays using =, ==, <, ...
  - Can't use scanf or printf to read or write an entire array
- But, you can do these things one element at a time.

“Parallel” Arrays

A set of arrays may be used in parallel when more than one piece of information must be stored for each item.

Example: we are keeping track of a group of students. For each item (student), we might have several pieces of information such as scores.

Parallel Arrays Example

Suppose we have a midterm grade, final exam grade, and average score for each student.
#define MT_WEIGHT 0.30
#define FINAL_WEIGHT 0.70
#define MAX_STUDENTS 200
int num_students,
    midterm[MAX_STUDENTS],
    final[MAX_STUDENTS];
double score[MAX_STUDENTS];
/* Suppose we know the number of students, have read student i’s grades for midterm and final, and stored them in midterm[i] and final[i]. Now:
Store a weighted average of exams in array score. */
for ( i = 0 ; i < num_students; i = i + 1 )  {
    score[i] = MT_WEIGHT * midterm[i] + FINAL_WEIGHT * final[i] ;
}

Individual array elements can be used as parameters, just like other simple variables.

Examples:
printf( "Last two are %f, %f", rain[5], rain[6] ) ;
draw_house( color[i], x[i], y[i], windows[i] ) ;
scanf( "%lf", &rain[0] ) ;
swap( &rain[i], &rain[i+1] ) ;

Array parameters (entire arrays) work differently:
An array is never copied (no call by value)
The array name is always treated as a pointer parameter
The & and * operators are not used

Programming issue: in C, arrays do not contain information about their size, so the size often needs to be passed as an additional parameter.

VECTOR SUM
/* Set vsum to sum of vectors a and b. */
void vectorSum( int a[3], int b[3], int vsum[3] )  {
    int i ;
    for ( i = 0 ; i < 3 ; i = i + 1 )
        vsum[i] = a[i] + b[i] ;
    }

#define ARRAY_SIZE 200

#include <stdio.h>

int main(void)  {
    int x[ARRAY_SIZE] = {1,2,3}, y[3] = {4,5,6}, z[3] ;
    vectorSum( x, y, z ) ;
    printf( "%d %d %d", z[0], z[1], z[2] ) ;
    return 0 ;
}

/* Set vsum to sum of vectors a and b. */
void vectorSum( int a[ARRAY_SIZE], double b[ARRAY_SIZE] )  {
    int i, total = 0 ;
    for ( i = 0 ; i < ARRAY_SIZE ; i = i + 1 )
        total = total + a[i] ;
    return ((double) total / (double) ARRAY_SIZE) ;
}
Unevent the size is omitted in an array parameter declaration.

```c
/* sum the vectors of the given length */
void vectorSum( int a[ ], int b[ ], int vsum[ ] ,
   int length) {
   int i ;
   for ( i = 0 ;  i < length ;  i = i + 1 )
      vsum[i] = a[i] + b[i] ;
}
```

```c
vectorSum( x , y , z , 3 );
```

### General Vector Sum

### Bonus Topic: Initializing Arrays

Review: "Initialization" means giving something a value for the first time.

General rule: variables have to be initialized before their value is used.

Review: Various ways of initializing assignment statement
scanf (or other function call using &)
initializer when declaring parameters (initialized with argument values)

### Array Initializers

```c
int w[4] = {1, 2, 30, -4};
   /*w has size 4, all 4 are initialized */
char vowels[6] = {'a', 'e', 'i', 'o', 'u'};
   /*vowels has size 6, only 5 have initializers */
   /*vowels[5] is uninitialized */
```

Caution: cannot use this notation in assignment statement:
```c
w = {1, 2, 30, -4}; /*SYNTAX ERROR */
```

Now the mystery of that midterm #1 question becomes clear!

### Incomplete Array Size

```c
double x[ ] = {1.0, 3.0, -15.0, 7.0, 9.0};
   /*x has size 5, all 5 are initialized */
```

But:
```c
double x[ ];    /* ILLEGAL */
```

### Summary

Arrays hold multiple values
All values are of the same type

Notation: [i ] selects one array element
[0] is always the first element
C does not check array bounds!

Especially useful with large amounts of data
Often processed within loops
Entire array can be passed as a parameter

### QOTD: (Un)Arrayed for Battle

Like pointers, arrays are dangerous programming constructs. Small mistakes can lead to disastrous and difficult to detect errors. Moreover, C will blithely stand by as you make many of these errors!

Which of these are errors according to C? Which are warnings in MSVC? Can you think of any other array dangers?

- int length = 5;
  - int elts[length];
  - int parallelArray1[SIZE], parallelArray2[SIZE];

- Set first elt of the parallel arrays to (2, 4). (?)
  - parallelArray1[0] = 2; parallelArray2[1] = 4;

- int elts[SIZE];
  - int elts[SIZE], numElts;
  - for (i = 0; i <= SIZE; i++) /* Initialize the array. */
    - elts[i] = 0;
    - numElts++;

  hint: is there a surefire way you can test these?
Parallel Arrays in HW4

What data in HW4 can be represented as a set of parallel arrays?

What is each "column"? What is each "row"?

What functions now need to have array parameters?
Find where those functions are called. How will the call be different?