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

Example: Rainfall Data

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?

Question of the day: Can we do all of these while we read the data?

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

The first of these (today's topic) is *arrays*

O-7

Arrays *(Think "Table")*

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

The entire collection is given a name

Each individual element is referred to by a number, called it's *index*

Example: rainfall for one week

double rain[7];

0	1.0
1	0.2
.	0.0
.	0.0
.	1.4
6	0.1
	0.0

O-8

Examples

```
double  rain[7];        // each element has rainfall
                        // on one day for this week
int     grades[200];    // midterm 1 scores

char    alphabet[26];   // 'a', 'b', …, 'z'
```

This allocate entire arrays and give each a name (rain, grades, and alphabet)

O-9

Accessing Individual Elements

Rainfall for one week

double rain[7];

1.0
0.2
0.0
0.0
1.4
0.1
0.0

Variable access:

rain[0] is 1.0

rain[6] is 0.0

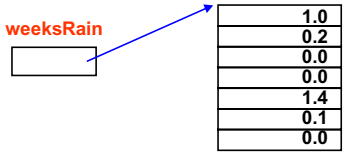
rain[4]*2.0 is 2.8

O-10

Under the Covers (of C…)

double weeksRain[7];

causes C to allocate the following:



There is no way in your program to say anything that means "all the elements"!

O-11

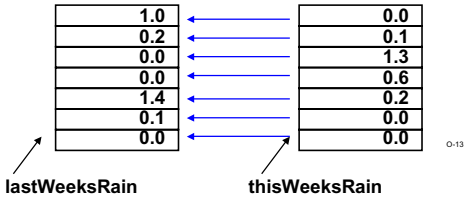
So What? (#1)

```
double thisWeeksRain[7];
double lastWeeksRain[7];

    lastWeeksRain = thisWeeksRain;    ILLEGAL

for (day=0; day<7; day++) {
    lastWeeksRain[day] = thisWeeksRain[day];
}
```

O-12

So What? (#1)

```
for (day=0; day<7; day++) {
    lastWeeksRain[day] = thisWeeksRain[day];
}
```



O-13

Array Declaration Syntax

type name[size]; ← array declaration

size must be an int constant expression

Known value at compile-time

```
double weeksRain[7];
double monthsRain[7*4];
double yearsRain[52*DAYS_PER_WEEK];
```

O-14

Symbolic Constants For Arrays

Yes!

You should never, ever, write literal constants for array bounds:

NO **double weeksRain[7];**

YES **double weeksRain[DAYS_PER_WEEK];**

BUT there's a catch…

O-15

Symbolic Constants For Arrays

Array bounds must be "constant expressions" that can be evaluated at compile-time.

```
int DAYS_PER_WEEK=7;
```
won't work (because it's really a variable, not a constant known at compile-time)

```
#define DAYS_PER_WEEK  7
```

O-16

Symbolic Constants For Arrays

```
// array bounds
#define     DAYS_PER_WEEK           7
// symbolic constants
double      TORRENT                 = 2.0;
int main(void)
{
   double     thisWeeksRain[DAYS_PER_WEEK];
   for (day=0; day<DAYS_PER_WEEK; day++) {
            ReadDouble(&thisWeeksRain[day]);
            if (thisWeeksRain[day] > TORRENT) {
                printf ("Day %d: Wow!\n", day);
            }
   }
   return 0;
}
```

O-17

#define

Doesn't follow ANY of the rules of C

```
#define DAYS_PER_WEEK   7
```

NOT A VARIABLE NO "=" NO ";"

Does "text substitution" BEFORE compile-time

Is an opportunity for errors that are impossible to debug

Use it only for array bounds!

Use it only with simple, integer, constants!

O-18

Array Terminology

double rain[DAYS_PER_WEEK];

rain is of type array of double with capacity DAYS_PER_WEEK.

rain[0], rain[1], ... are the elements of the array rain.

Each element is a variable of type double.

0,1, ... , DAYS_PER_WEEK-1 are the indices of the array. Also called subscripts.

The bounds are the lowest and highest values of the subscripts (here: 0 and DAYS_PER_WEEK-1).

O-19

Array Elements as Parameters

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

```
printf( "Last two are %f, %f", rain[5], rain[6] ) ;

DrawHouse( color[i], x[i], y[i], windows[i] ) ;

scanf( "%lf",  &rain[0] ) ;

swap( &rain[i], &rain[i+1] ) ;
```

O-20

Whole Arrays as Parameters

Array parameters (entire arrays) work differently:

- An entire array is never copied
- 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.

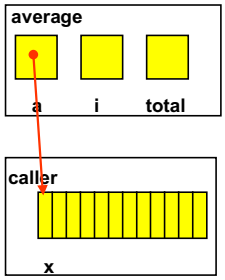
O-21

Array Parameter Example

```
#define ARRAY_SIZE  200
double Average ( int myArray[ARRAY_SIZE] ) {
    int index, total = 0 ;
    for (index = 0 ; index < ARRAY_SIZE ; index++ ) {
       total = total + myArray[index] ;
    }
    return ((double) total /  (double) ARRAY_SIZE) ;
}

int x[ARRAY_SIZE] ;
...
xAvg = Average ( x ) ;
```

O-22

Picture




```
#define ARRAY_SIZE  200
double Average (
            int a[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) ;
}

int x[ARRAY_SIZE] ;
...
xAvg = Average ( x ) ;
```

O-23

Vector Sum Example

```
/* 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] ;
}

int main(void) {
    int x[3],  y[3], z[3] ;
    ...
    VectorSum( x , y , z );
    printf( "%d %d %d", z[0], z[1], z[2] ) ;
}
```

note: no * no &

O-24

General Vector Sum

Usually the size is omitted in an array parameter declaration.

```
/* sum the vectors of the given length */
void VectorSum( int a[ ] , int b[ ] , int vsum[ ] ,
                                int length)  {
    int index ;
    for (index = 0 ; index < length ; index++ )
       vsum[index ] = a[index ] + b[index ] ;
}
…
int x[7],  y[7],  z[7] ;
VectorSum( x , y , z , 7 );
```

O-25

Rainfall Analysis (cont.)

Strategy for processing data if we need all of it before we can process it:

- Read data and store it in an array
- 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.

O-26

Keeping Track of Elements In-Use

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

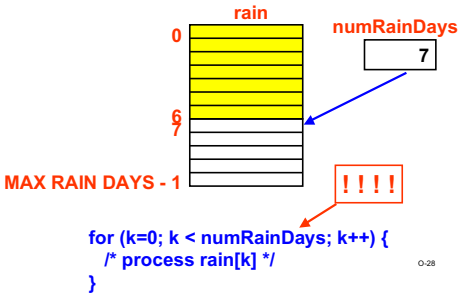
```
#define MAXRAINDAYS 400
int     rain[MAXRAINDAYS];
```

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 – the current *size* of the list – in a separate variable

O-27

Keep the valid entries together



O-28

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

O-29

Declarations

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

O-30

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);
}
```

O-31

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;
```

O-32

We should add 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++;
}
printf("%d days above the average of %.3f.\n"
            numAbove, rainAverage);
```

O-33

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 ≤i+3+k ≤6*

The index may be very simple

rain[0]

or incredibly complex

rain[(int) (3.1 * fabs(sin (2.0*PI*sqrt(29.067))))]

(although too complex is confusing and bad style)

O-34

C Array Bounds are Not Checked

```
#define DAYS_IN_WEEK 7

double rain[DAYS_IN_WEEK] ;
int index ;
index = 900 ;
...
rain[index ] = 3.5 ;
```

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.

O-35

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.

O-36

"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

O-37

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] ;
```

O-38

Parallel Arrays Example

```
/*  Suppose we know the value of  num_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 for each
    student in array score.  */
for ( i = 0 ;   i < num_students;   i = i + 1 )  {
        score[i] = MT_WEIGHT * midterm[i] +
                   FINAL_WEIGHT * final[i] ;
}
```

O-39

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)

O-40

Array Initializers

```
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:

```
w = {1, 2, 30, -4};      /*SYNTAX ERROR */
```

O-41

Incomplete Array Size

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

But:

```
double x[ ];                /* ILLEGAL */
```

O-42

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