Arrays

Chapter 8

Objectives

To be able to:

- Effectively use arrays to hold ordered collections of similar data.
- Correctly access array entries with C statements.
- Avoid reading and writing nonexistent entries beyond the end of an array.
  - Understand the symptoms of doing so.

Arrays

- Arrays are a way of defining a lot of essentially identical variables without having to give a unique name to each.
  - Like a table.
  - Identify a specific element by its position.
  - Convenient to process with a loop.

Declaring an Array

- Declare an array like a normal variable followed by square brackets:

```
int numbers[5];
```

- This is an array of five integers.
- Similar to

```
int n0;
int n1;
int n2;
int n3;
int n4;
```

Referring to an Array Entry

- We refer to an array entry by specifying an *index* value after the array name.

```
n[3] = 1234;
```

- Index values in C always start with 0.
- So this is the fourth entry in array n.

Referring to an Array Entry

- An array name followed by an index value is equivalent to a variable of the same type.
  - Works on either side of =
- Examples:
  - n[3] = 1234;
  - value = n[3]; // value is an int variable.

Average and Deviations

```
#include <stdio.h>

int main()
{
    int x = 0;
    int y = 0;
    int z = 0;
    double sum = 0.0, average = 0.0;

    printf ("This program determines the average value\n");
    printf ("of three integers, x, y, and z, that you\n");
    printf ("enter from the keyboard.\n"
    printf("It then prints the average and the difference\n");
    printf("between the average and each value\n");

    printf ("x: ");
    scanf ("%d", &x);
    printf ("y: ");
    scanf ("%lg", &y);
    printf ("z: ");
    scanf ("%d", &z);
```

```
    sum = x + y + z;
    average = sum/3;

    printf("Sum of numbers is %f\n\n",sum)
    printf("The average is %f\n",average);
    printf("Numbers and differences from average\n");
    printf("%6d %6.2f\n",x,x-average);
    printf("%6df %6.2f\n",y,y-average);
    printf("%6df %6.2f\n",z,z-average);
    printf("\n");
}
```

The “Average and Deviations” Program

- What if we had wanted the average and deviations of 10 numbers rather than 3?
  - or 100?
  - or 1000?
- Arrays provide a convenient solution.
  - Read the numbers into an array, and retain the number of value entered in a variable.
  - Compute the sum and average
  - Print the deviations for each value

Program sum_and_dev.c

```
#include <stdio.h>
#define MAXSIZE 25

int main()
{
    int a = 0, i = 0, count = 0, sum = 0;
    double average = 0.0;
    int num[MAXSIZE];

    printf("This program computes the sum, average and deviations from the ");
    printf("average for up to %d values\n",MAXSIZE);\
    printf("Enter nonzero values ending with an entry of zero\n\n");

    printf("Enter next integer, 0 to quit: ");
    scanf("%d",&a);
    while( count < MAXSIZE && a != 0)
    {
        num[count] = a;
        count++;
        sum += a;
        printf("Enter next integer, 0 to quit: ");
        scanf("%d", &a);
    }
```

Program sum_and_dev.c

```
    if (count == 0)
    {
        printf ("No numbers entered\n");
    }
    else
    {
        printf("Number of values entered: %d\n",count);
        printf("Sum of values: %d\n",sum);
        average = (double) sum / count;
        printf("The average is %3.2f\n",average);
        printf("\nNumbers and differences from average\n");
        for (i = 0; i < count; i++)
        {
           printf("%6d %6.2f\n",num[i],num[i]-average);
        }
    }
   return 0;
}
```

Run middle_value.c

```
-bash-3.00$ gcc -Wall sum_and_dev.c
-bash-3.00$ ./a.out
This program computes the sum, average and deviations from the average for up to 25 values
Enter nonzero values ending with and entry of zero

Enter next integer, 0 to quit: 2
Enter next integer, 0 to quit: 7
Enter next integer, 0 to quit: 5
Enter next integer, 0 to quit: 3
Enter next integer, 0 to quit: 0
Number of values entered: 4
Sum of values: 17
The average is 4.25

Numbers and differences from average
     2  -2.25
     7   2.75
     5   0.75
     3  -1.25
-bash-3.00$
```

Accessing Array Elements

- We specify the *address* of an array element just like we do for a normal variable:

```
scanf ("%d", &num[i]);
```

Address of a[i]

Accessing Array Elements

- We normally use a *variable* to specify which entry of an array we want to use

  Example

  ```
  printf ("%d:  %d\n", i, num[i]);
  ```

  i is called the *index* value
  or, sometimes, the *subscript*

  where **i** is a variable of type int.

- Array always starts with entry 0.
- Last index value is (the length of the array) – 1.

- Note that the index is always an integer, regardless of the data type of the array itself.

Loops

- We often use a *loop* to access each element of an array in turn:

```
for (i = 0; i < MAXSIZE; i++)
{
    printf ("%d:  %d\n", i,num[i]);
}
```

Standard form to step through an array.

- The “for” loop is especially well suited for indexing through an array.
  - “For each element of n, do the following:”

Things to Notice

Start with element 0

(First entry in the array)

Length of the array

Step to next element

```
for (i = 0; i < MAXSIZE; i++)
{
   printf ("%d:  %lg\n", i, a[i]);
}
```

Programming Style Issue

- It is common to use variables such as i, j, and k as array indexes.
- This is an acceptable use of a single character variable name.
  - Similar to use of subscripts in mathematical expressions.
  - Readers typically will recognize these variables as array indexes.
  - They usually have no inherent meaning.

A Common Mistake

```
double n[10];

...

for (i = 1; i <= 10; i++)
{
    printf ("Next number: ");
    scanf ("%lg", &n[i]);
}
```

Incorrect!

What happens when i is 10?

scanf reads into n[10]

But there **is no** entry 10 for array n

A Common Mistake

- When you write past the end of an array you trash some piece of your program's memory.
  - Program may crash.
  - Nothing bad may happen.
  - A problem may show up much later.
- C does not protect you from this kind of error.
- It's up to you to be sure your array index values are valid.

A Good Test Question

Consider the following program fragment:

```
double n[10];
int i;

...

for (i = 1; i <= 10; i++)
{
    printf ("Next number: ");
    scanf ("%lg", &n[i]);
}
```

What's wrong with this program?

Array Initializers

Just as you can initialize a variable at compile time

```
int age = 21;
```

you can initialize an array at compile time

```
int age[5] = {21, 25, 28, 32, 45};
```

Note curly brackets

Array Initializers

If you provide an initializer, you can omit the array size:

```
int age[] = {21, 25, 28, 32, 45};
```

The compiler determines the array size from the number of values in the initializer.

Array Initializers

- What if a size is specified and an initializer is given, and the sizes do not match?
- Too many values in the initializer
  - Compile time error
- Not enough values in the initializer
  - Remaining entries are set to 0.
- Set the entire array to 0's by providing empty brackets for the initializer:

```
int instock[6] = {0 }; // = {} ???
```


End of Section

An Array Overrun

```
#include <stdio.h>

int main()
{
    int i;
    int a[10] = {0,1,2,3,4,5,6,7,8,9};

    for (i = 1; i <= 12; i++)
    {
        printf ("i = %d  a[%d] = %d   \n", i, i, a[i]);
        a[i] = 0;
    }
    return 0;
}
```

Program Running on Windows

c:\Documents and Settings\Rollins\My Documents\Visual Studio 2005\Projects\Program_Desi...

```
i = 1  a[1] = 1
i = 2  a[2] = 2
i = 3  a[3] = 3
i = 4  a[4] = 4
i = 5  a[5] = 5
i = 6  a[6] = 6
i = 7  a[7] = 7
i = 8  a[8] = 8
i = 9  a[9] = 9
i = 10  a[10] = -858993460
i = 11  a[11] = -858993460
i = 12  a[12] = 12
i = 1  a[1] = 0
i = 2  a[2] = 0
i = 3  a[3] = 0
i = 4  a[4] = 0
i = 5  a[5] = 0
i = 6  a[6] = 0
i = 7  a[7] = 0
i = 8  a[8] = 0
i = 9  a[9] = 0
i = 10  a[10] = 0
i = 11  a[11] = 0
i = 12  a[12] = 12
i = 1  a[1] = 0
i = 2  a[2] = 0
i = 3  a[3] = 0
```

What’s going on here?

Program Running on Unix

sunblast.eng.usf.edu - [default] - F-Secure SSH Client

File Edit View Window Help

Quick Connect Profiles

```
233 Sun Server:~/test:
233 Sun Server:~/test:
233 Sun Server:~/test: gcc -Wall demo.c
234 Sun Server:~/test: a.out
i = 1  a[1] = 1
i = 2  a[2] = 2
i = 3  a[3] = 3
i = 4  a[4] = 4
i = 5  a[5] = 5
i = 6  a[6] = 6
i = 7  a[7] = 7
i = 8  a[8] = 8
i = 9  a[9] = 9
i = 10  a[10] = 4
i = 11  a[11] = 11
i = 1  a[1] = 0
i = 2  a[2] = 0
i = 3  a[3] = 0
i = 4  a[4] = 0
i = 5  a[5] = 0
i = 6  a[6] = 0
i = 7  a[7] = 0
i = 8  a[8] = 0
i = 9  a[9] = 0
i = 10  a[10] = 0
i = 11  a[11] = 11
i = 1  a[1] = 0
i = 2  a[2] = 0
```

Connected to sunblast.eng.usf.edu | SSH2 - 3des-cbc - hmac-sha1 - none | 48x28 | 17, 35 | 0

Slowing Down an Endless Loop

- Try a google search for “delay in C program”
- On Windows, #include <windows.h> and use **`Sleep(ms);`**
- On Linux, use **`sleep(sec);`**
- Both are system dependent functions.
  - Do not use in projects for this class!

An Array Overrun

- On Windows

  a[12] is i

- On Unix

  a[11] is i

Invalid Array References

- Whenever you find a program behaving strangely --
  - e.g., Value of a variable mysteriously changes when you haven't touched it

    suspect a bad index value for an array reference.

- No easy way to track down.
  - You have to check line by line.
  - Error may be nowhere near the place where the symptom appears.
  - Will learn some defensive code shortly.

End of Section

Array Types

- Any type that you can use for a variable can also be used for an array.

```
int age_in_years [25];

double weight [100];

char name [30];
```

Base Type

Array *Length*

The *index* is always an integer type, regardless of the base type.

C Terminology

- In C, *Length* is not the same thing as *size*.
  - Length is number of elements.
  - Size is number of bytes occupied.
- Size is Length * (Size of one element)
- You can compute the length of an array n as:

```
sizeof(n) / sizeof(n[0])
```

Works for arrays of any base type

but only where the declaration is in scope.

Size vs. Length

```
[turnerr@login4 test]$ cat array_size.c
#include <stdio.h>

int main( void )
{
    double n[10];

    printf ("sizeof n is %d\n", (int) sizeof(n));
    printf ("sizeof n[0] is %d\n", (int) sizeof(n[0]));
    printf ("Length of n is %d\n", (int) sizeof(n) / (int) sizeof(n[0]));
    return 0;
}

[turnerr@login4 test]$ gcc -Wall array_size.c
[turnerr@login4 test]$ ./a.out
sizeof n is 80
sizeof n[0] is 8
Length of n is 10
[turnerr@login4 test]$
```

Invalid Array References

- One safeguard against bad array references:
  - Use **`assert`** to check index value.

Invalid Array References

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <assert.h>

int main()
{
    int i = 0;
    double n[3];
    for (i = 1; i <= 3; i++)
    {
        printf ("Next number: ");
        assert ( i < (sizeof(n)/sizeof(n[0])));
        scanf ("%lg", &n[i]);
    }
    return 0;
}
```

Good programming practice: Always verify that array index is valid.

Assertion Failure on Unix

```
208 Sun Server:~/test: gcc -Wall array_limit_example.c
209 Sun Server:~/test: a.out
Next number: 1
Next number: 2
array_limit_example.c:11: failed assertion `i < (sizeof(n)/sizeof(n[0]))'
Next number: Abort
210 Sun Server:~/test:
210 Sun Server:~/test:
```

File name and line number

The assertion that failed.

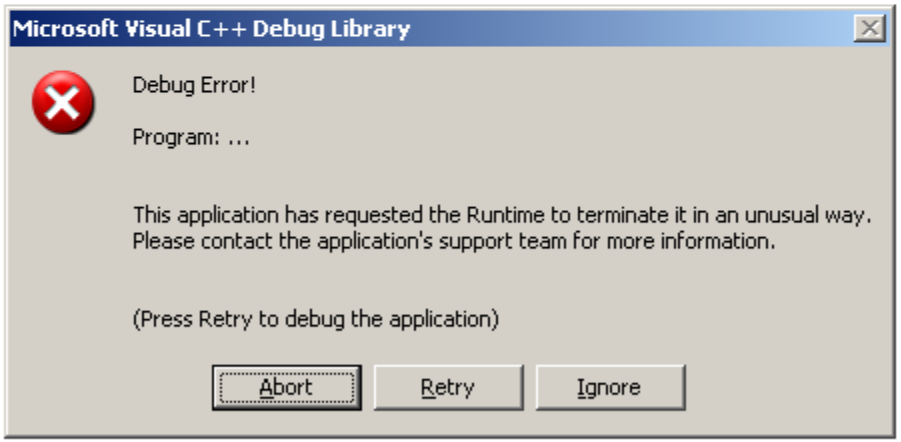
Assertion Failure on Windows

c:\Documents and Settings\Rollins\My Documents\Visual Studio 2005\Projects\Program_Design_2008_Fall\demo\demo\Debug\demo.e...

```
Next number: 1
Next number: 2
Next number: Assertion failed: i < (sizeof(n)/sizeof(n[0])),
 file c:\documents and settings\rollins\my documents\visual
studio 2005\projects\program_design_2008_fall\demo\demo\demo
.c, line 11
```

The assertion that failed.

File name and line number



Multidimensional Arrays

- Arrays can have any number of dimensions
- A two-dimensional array is just an array of arrays
- Declaration: `int m[5][9];`
- Array m above has 5 rows and 9 columns
- Like a 5x9 matrix in mathematics
- Both rows and columns are indexed starting at 0
- The entry in the ith row and j^{th} column is `m[i][j]`
- **DON'T WRITE m[i,j] – COMPILER WILL OBJECT**!

Multidimensional Arrays in Memory

- Declaration: `int m[5][9];`
- Array m above has 5 rows and 9 columns
- Like a 5x9 matrix in mathematics
- Both rows and columns are indexed starting at 0
- The entry in the ith row and j^{th} column is `m[i][j]`
- **DON'T WRITE m[i,j] – COMPILER WILL OBJECT**!

Multidimensional Array Initialization

```
int m[5][9] = {{1,1,1,1,1,1,1,1,1},
               {0,1,0,1,0,1,0,1,0},
               {1,1,1,0,0,0,1,1,1},
               {0,0,0,0,0,0,0,0,0} };
```

Inner braces may be omitted

Summary

- Arrays are an essential part of C.
- Most real world C programs use arrays extensively.
- Array entries are essentially identical to individual variables.
  - Use them in the same ways
  - Index value specifies which element is used.
- Be *very careful* not to use an index beyond the end of the array.

Assignment

- Read Chapter 8
- Type in, compile, and run the examples from this lecture.
  - Be sure you understand what is happening.
  - Ask for help if you don’t.