Arrays

Darin Brezeale

The University of Texas at Arlington

Arrays

While single variables have many uses, there are times in which we wish to store multiple related values. For these we may wish to use an array.

The mathematical equivalent of a one dimensional array is a vector.

Example: We may have the vector $X = (3.5, 4.0, 9.34)$ whose terms are referenced as x_1, x_2, and x_3.

Arrays – Declaring

The declaration for arrays is similar to the declaration for other data types. The difference is the addition of square brackets (i.e., []) to the declaration.

Example: If we had a single variable of type `int`, we would declare it using

```
    int some_variable;
```

To create an array of 10 variables of type `int`, we would declare it using

```
    int some_array[10];
```

Arrays – Declaring cont.

Example of declaring and initializing an array.

```
double someData[3];   /* declare the array someData
                         that will hold 3 doubles */

/* later we can provide the specific array values.
   notice how the array index begins at 0 */
someData[0] = 3.5;
someData[1] = 4.0;
someData[2] = 9.34;
```

Here we declare and initialize the array at the same time, so we don’t need to include the number of array members in the square brackets.

```
double myData[] = {3.5, 4.0, 9.34};
```

Arrays – Example

Once the array has been initialized, the members can be referenced using the array name and the index of the member.

```
#include <stdio.h>

int main(void)
{
    double myData[] = {3.5, 4.0, 9.34};

    printf("The last member of myData is %3.2f\n", myData[2]);

    myData[2] = 6;
    printf("The last member of myData is now %3.2f\n", myData[2]);
}
```

Output

```
    The last member of myData is 9.34
    The last member of myData is now 6.00
```

Arrays – Notes

Some comments on using arrays:

- Each element of the array will be of the same type.
- Array indices began at 0.
- WARNING: A common error is the “array out of bounds” error that occurs when the index goes beyond the declared size.

Arrays – Indexing Error

This program attempts to change a value outside the assigned range.

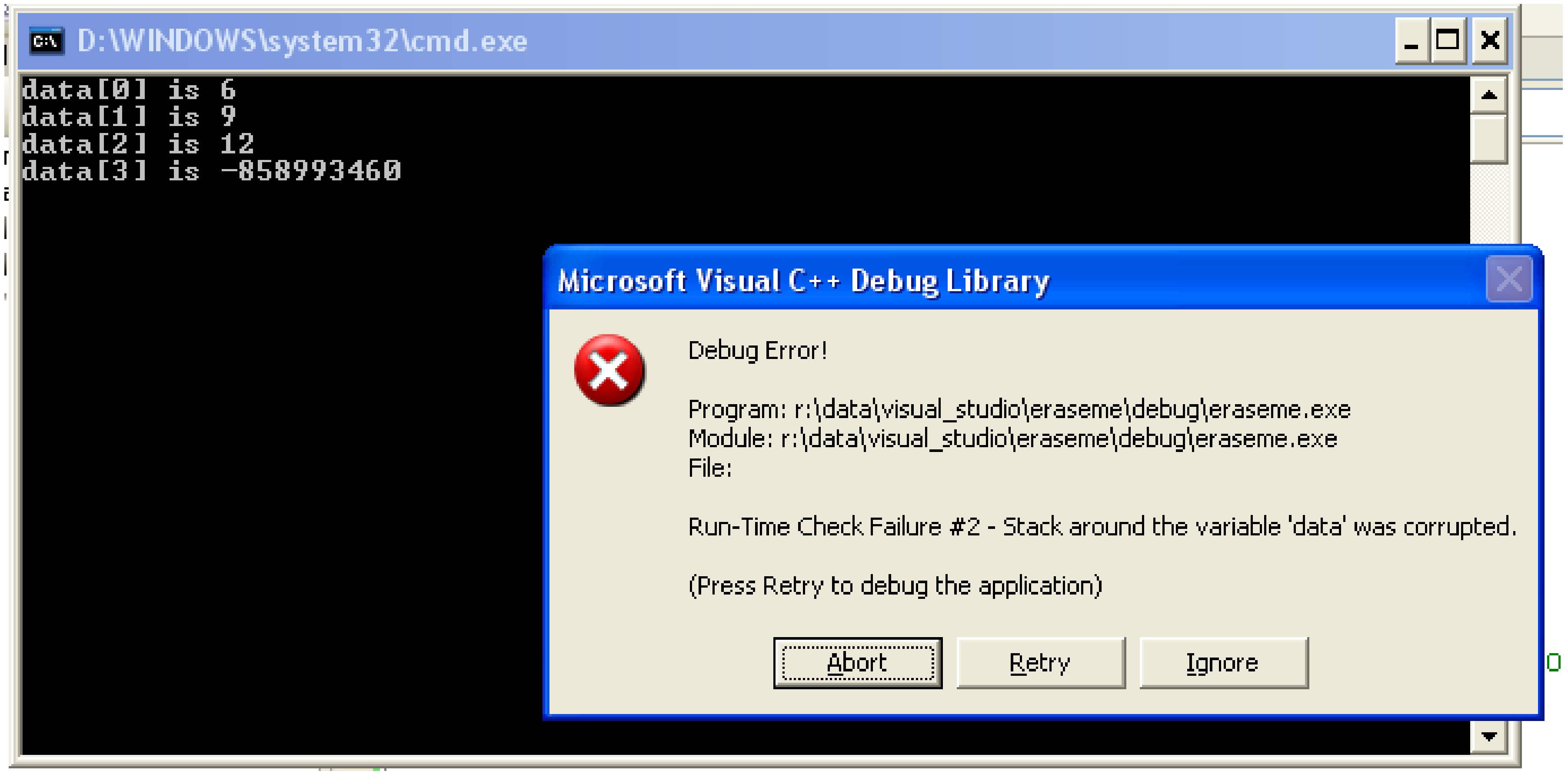
```
#include <stdio.h>

int main(void)
{
    int data[3] = {6, 9, 12};  /* these are referenced as data[0],
                                  data[1], and data[2]             */
    int i;

    /* index stops at 3 instead of 2 as it should */
    for(i = 0; i <= 3; i++)
        printf("data[%d] is %d\n", i, data[i]);

    data[3] = 8;  /* here is where the problem occurs */
}
*/
```

Arrays – Indexing Error cont.


D:\WINDOWS\system32\cmd.exe

```
data[0] is 6
data[1] is 9
data[2] is 12
data[3] is -858993460
```

Microsoft Visual C++ Debug Library

Debug Error!

Program: r:\data\visual_studio\eraseme\debug\eraseme.exe
Module: r:\data\visual_studio\eraseme\debug\eraseme.exe
File:

Run-Time Check Failure #2 - Stack around the variable 'data' was corrupted.

(Press Retry to debug the application)

More 1D Array Examples

```
#include <stdio.h>

int main(void)
{
    /* reserve memory for 4 ints; initialize the first 2 here */
    int data[4] = {6, 94};  /* data[0] and data[1]          */
    int i;

    data[3] = 3;

    for(i = 0; i < 4; i++)
        printf("data[%d] is %d\n", i, data[i]);
}
```

Output

```
data[0] is 6
data[1] is 94
data[2] is 0
data[3] is 3
```

Initialization Notes

If we initialize only some of the array values when the array is declared,

- The initialized values will be at the beginning of the array.
- The remaining values will be initialized to zero.

See `example-array-initialize.c` on the course website.

Arrays versus Single Variables

Why should we use arrays when we could just use single variables? When we have many related values, storing them in an array can make them easier to work with.

See `example-array_vs_var.c` on the course website.

Arrays and Functions

We can pass arrays to functions just as we do with other variable types.

Example: The definition for a function that receives an array and returns a double.

```
    double some_function(int data[])
```

See `example-array-function.c` on the course website.

Arrays and Functions cont.

There is a significant difference between passing variables to functions and passing arrays to functions.

- When passing a variable by value, a copy of the variable is used in the function and changes to it do not affect the original.
- When passing an array to a function, we are actually passing the address of the original so changes to the array within the function DO affect the original.

See `example-array-function2.c` on the course website.

Multidimensional Arrays

So far we have dealt with one-dimensional arrays. We can also have arrays of arrays, also known as multidimensional arrays.

A two-dimensional array is really two one-dimensional arrays.

The basic form for declaring a 2D array is

```
    type array_name[rows][columns];
```

The mathematical equivalent of a two-dimensional array is a matrix.

Multidimensional Arrays cont.

A 2D array can be viewed like a table.

1	2	3
4	5	6

To create a 2D array for this table of values, we could use

```
    int some_data[2][3] = { {1, 2, 3},
                            {4, 5, 6} };
```

Multidimensional Arrays cont.

As with one-dimensional arrays, 2D arrays indices begin at zero. The difference is now we have to keep track of indices in two dimensions. So, if we create the following array:

```
    int sales[2][3] = { {1, 2, 3},
                        {4, 5, 6} };
```

the individual elements are referenced with the following combinations of array name and indices:

sales[0][0]	sales[0][1]	sales[0][2]
sales[1][0]	sales[1][1]	sales[1][2]

2D Array Example

```
#include <stdio.h>

int main(void)
{
    int i;
                      /* age, weight (lbs), height (in) */
    int demographics[2][3] = { {24, 180, 72},
                               {39, 175, 65} };

    /*  print the weights, which are in the second column.
        Remember, the second column has an index of 1.    */
    for(i = 0; i < 2; i++)
        printf("%d\n", demographics[i][1]);
}
```

Output

```
180
175
```

3D Array Example

```c
#include <stdio.h>
int main(void)
{
    /* two 2D arrays; each 2D array consists of three 1D arrays
       with four elements */
    int data[2][3][4] = {{{ 1,  2,  3,  4},
                          { 5,  6,  7,  8},
                          { 9, 10, 11, 12}},
                         {{13, 14, 15, 16},
                          {17, 18, 19, 20},
                          {21, 22, 23, 24}}};
    int i;

    /* print the second row of the second 2D array */
    for(i = 0; i < 4; i++)
        printf("%d ", data[1][1][i]);
}
```

Output

```
    17 18 19 20
```

1D and 2D Arrays Differences

When declaring a 2D array, the number of columns must be stated.

Example

```
int array1D[] = {1, 2, 3};

int array2D[][3] = { {4, 5, 6},
                     {7, 8, 9} };
```

1D and 2D Arrays Differences

In function declarations and definitions that have 2D arrays as parameters, the number of columns must be stated.

```
#include <stdio.h>

void printColumn(int, int [][3]); /* function declaration */

int main(void)
{
    int array2D[][3] = { {4, 5, 6},
                         {7, 8, 9} };
    printColumn(2, array2D); /* print the third column */
}

void printColumn(int column, int input[][3])
{
    int i;
    for(i = 0; i < 2; i++)
        printf("%d\n", input[i][column]);
}
```