Arrays as Function Parameters

Arrays as Function Parameters

- We can pass an array to a function
  - Important difference from passing an individual variable (double, int, etc.)

- When we pass an individual variable to a function, the *value* of the variable is passed to the function.
  - The function gets a *copy* of the variable, not the actual variable.
  - A parameter value is set to the value of the argument.
  - Parameter acts like a local variable in the function.
  - Changes to the parameter have no effect on the variable used in the function call.

Arrays as Function Parameters

- When we pass an array to a function, the *address* of the array is passed to the function.
  - A parameter value is set to the address of the array used as an argument.
  - The function has access to the array that was used as an argument.
    - Not a copy!
  - Any changes made by the function are made to the original array.

Arrays as Function Parameters

- An array parameter lets a function pass multiple values back to the caller
  - in the caller’s array.

- We cannot use an array as the returned value for a function.
  - Returned value must be a single variable.

Length of Array Argument

- When we use an array as a function parameter, only the *address* of the array is passed to the function
  - Not the *length*.
- Function must learn length of the array by some other means.

Length of Array Argument

- Function must learn length of the array by some other means.
  - Hard coded value in definition and in function.
    - Bad technique
    - Hazard for future changes
  - #define used for length in array declaration.
    - Better, but ...
    - Hidden coupling -- *not visible in the function signature*.
  - “Sentinel”
    - Special value that can never appear as data indicates end of array (or end of data)
  - Additional parameter specifies length of the array
    - Better solution!

Example

- Compute the average for an array of real numbers.
  - Put the calculation in a function and pass the array to the function.
  - Function returns the average.

program array_ave_func.c

```
double array_average (double x[], int length )
{
    double sum =0.0;
    double average = 0.0;
    int i = 0;

    assert (length > 0);
    /* Compute the average. */

    for (i = 0; i < length; i++)
    {
        sum += x[i];
    }
    average = sum / length;
    return average;
}
```

Caller must specify the length of the array used as first argument.

Square brackets say that x is an array.

Note that the brackets are empty.

Use length passed by caller to terminate loop

Array parameter is used exactly like an array declared locally.

Divide by length passed by caller

program array_ave_func.c

```
int main( void )
{
    double numbers[10];
    double ave;
    int i;

    printf( "\nCompute the average of 10 numbers\n" );

    /* Get the numbers to be averaged. */
    printf( "Please input 10 real numbers: \n" );
    for (i = 0; i < 10; i++)
    {
        printf ("Next number: ");
        scanf ("%lg", &numbers[i]);
    }

    ave = array_average( numbers, 10);

    printf( "The average is %g\n\n", ave );
    return 0;
}
```

Call the function passing array n and its length as arguments

Same as an individual variable (no "&")

Arrays as Function Parameters

- The function does not have to use the entire array as originally declared.
  - Caller can pass just part of an array to the function.
    - Specify a length less than the actual length.

- Let’s modify program array_ave.c to permit the user to enter any number of values up to 10.
  - Limit user to positive values.
  - Use negative value to indicate finished.

program array_ave_func2.c

```
double array_average (double x[], int length )
{
    double sum;
    double average;
    int i;
    printf ("Computing average of %d numbers\n", length);

    /* Compute the average. */
    sum = 0.0;
    for (i = 0; i < length; i++)
    {
        sum += x[i];
    }
    average = sum / length;
    return average;
}
```

New debugging message.

program array_ave_func2.c

```
int main( void )
{
    double n[10];
    double ave;
    int i;
    printf( "\nCompute the average of up to 10 positive real numbers\n" );

    /* Get the numbers to be averaged. */
    printf( "Please input up to 10 postive real numbers.
  \n" );
    printf( "Enter a negative value to terminate input.\n");
```

```
for (i = 0; i < 10; i++)
{
    double next_value;
    printf ("Next input: ");
    scanf ("%lg", &next_value);
    if (next_value < 0.0)
    {
        break;
    }
    n[i] = next_value;
}
```

Local to this code block

Read into local variable

Break out of loop if user inputs a negative value

Put user's input into next element of array n.

```
/* Value of i is number of array entries
   filled with postive values. */
ave = array_average(n, i);
printf( "The average is %g\n\n", ave );
return 0;
}
```

An important comment.

Pass number of elements used as array length

Run program array_ave_func2.c

```
turnerr@login1:~/test
[turnerr@login1 test]$
[turnerr@login1 test]$ gcc -Wall array_ave_func2.c
[turnerr@login1 test]$ ./a.out

Compute the average of up to 10 positive real numbers
Please input up to 10 postive real numbers.
Enter a negative value to terminate input.
Next input: 1
Next input: 1.5
Next input: 2
Next input: -1
Computing average of 3 numbers
The average is 1.5

[turnerr@login1 test]$
```

Modifying the Caller's Array

- Let's change array_ave_func2.c to use a function to *get* the user's inputs.
  - Function fills an array passed by the caller.
  - Return number of elements filled.

program array_ave_func3.c

```
/* Get up to 10 positive real numbers from user.
   Put values into caller's array.
   Argument "length" is length of caller's array.
   Return number of elements filled as function's value. */
int get_input (double q[], int length)
{
    int i;
    printf("Please input up to 10 postive real numbers. \n");
    printf("Enter a negative value to terminate input.\n");
    for (i = 0; i < length; i++)
    {
        double next_value;
        printf ("Next input: ");
        scanf ("%lg", &next_value);
        if (next_value < 0.0)
        {
            break;
        }
        q[i] = next_value;
    }
    /* Value of i is number of array entries filled with postive values.*/
    return i;
}
```

Function to compute average is same as before.

```
double array_average (double x[], int length )
{
    double sum;
    double average;
    int i;
    printf ("Computing average of %d numbers\n", length);

    /* Compute the average. */
    sum = 0.0;
    for (i = 0; i < length; i++)
    {
        sum += x[i];
    }
    average = sum / length;
    return average;
}
```

```
int main( void )
{
    double n[10];
    int number_of_values;
    double ave;
    printf("\nCompute the average of up to 10 "
           "positive real numbers\n");

    /* Get the numbers to be averaged. */
    number_of_values = get_input(n, 10);

    ave = array_average(n, number_of_values);
    printf("The average is %g\n\n", ave);
    return 0;
}
```

Multidimensional Array Parameters

- When passing a multidimensional array as a parameter, only the length of the first dimension may be omitted.

- Thus, a function to add two 2-dimensional arrays as matrices would have to pass in the number of columns:

```
void MatrixAdd(int A[][10], int B[][10],int Sum[][10])
{  // Places the sum of A and B into Sum
       …
```

Summary

- Arrays can be passed to functions
  - It's the *address* of the array that is passed to a function.
  - Not a copy, as with individual variables.
- A function with an array parameter accesses the caller's array.
- When writing a function that takes an array as a parameter, use a second parameter to specify the length of the array.