MECH 215

Lecture 4

Arrays

Arrays

- Arrays use a single name to refer to a collection of memory addresses that can be access by the name of the array and an index value.
- For example: an array called: my_array[3] will have three locations in memory (the size of each location is based on the data type stored in it. To access the first location and put the data into a variable we could write my_variable=my_array[0].
- To assign the value from a variable to an array we would write: my_array[0]=my_variable.
- The index value within the square brackets is used to access the different locations within the array.

Arrays

- The declaration of an array requires three things.
  - What the array is to store (int, char, long double, etc…)
  - The name of the array (any non-reserved word)
  - How many locations within the array (number of locations)
  - E.g.: int my_array_whatever[10]
- In C++ the default starting point of the index is zero. Why? The array name has a single value that is a location in memory that holds the starting location of the array. The index is the offset from the starting location in memory that you wish to access.
- Size of each location depends upon what is stored there. This depends upon the system you are using:

Arrays

- In order to determine the size of the data type you can use the sizeof function. For example to find the size of an integer you could write sizeof(int) in your program and then print it out.
  - For the linux system we have:
  - Char has a size of 1 byte (8 bits) default size 1
  - Short has a size of 2 bytes
  - Int has a size of 4 bytes
  - Long has a size of 4 bytes
  - Unsigned char has a size of 1 byte
  - Unsigned short has a size of 2 bytes
  - Unsigned int has a size of 4 bytes
  - Unsigned long has a size of 4 bytes
  - Float has a size of 4 bytes
  - Double has a size of 8 bytes
  - Long double has a size of 16 bytes
  - Void has a size of 4 bytes
- So if an array of double with 10 locations was declared the computer would have to reserve 80 bytes of memory for use.

Strings

- Strings are arrays of characters. For the most part they work the same as other arrays but they do have some special features.
- There are special libraries for strings:
  - #include<cctype> library will check to see if a character is upper or lower case, a black space, a letter or a number as well as other features.
  - #include<iomanip> this is used for formatting output to the screen or file. Works with number and letters. Can set width and precision of displayed values.
  - #include<cstring> library is used to check the length of a string, merge strings, copy strings (entirely rather than element by element), etc…
- Strings end with the \0 the null terminator which takes up one memory location within the string.

Strings and Arrays

- Strings are a class, this will be discussed later where the pointer (also discussed later), is the string name.
- While useful for many applications I will focus for now on arrays hold numerical values.
- When calling a function the array can be used as input to the function in a manner similar to that of a simple variable, the function declaration will have empty square brackets for a single dimension array and will have one empty square and all other brackets will have a constant value within them for multi-dimensional arrays.

Functions Using Arrays

- When a function accesses an array, the function has access to the original memory location of the array it does not make a copy. This is important since this means that the old data in the array can be overwritten and lost. This is an important feature of arrays, they are called by reference not by value (or parameter).
- This also means that if you intend to use a function to write to an array by returning a value from the function, you must call the function multiple times since a function can return only one value, and you can access only one location within an array at a time.

Multi-Dimensional Arrays

- It is sometimes helpful to use an array with more than one index. While it is helpful to think of the indices in terms of matrix forms so that the two dimensional array two_dim_matrix[row][col] will have the first index refer to the row number and the second index refer to the column number, this is more for you convenience.
- When using more than one array all but the first index must be specified when calling functions and you still can only access one location within the array at a time so all index values are required to read or write to an array.

Array In memory

- How 2-d arrays are stored in memory : Arrays will be “stacked” one row upon the other within memory.
- Main memory in a computer is accessed by the compiler as a linear sequence of bytes (address is like a street you must move along the street length until you arrive at the desired address).
- Hence, a two-dimensional array must be mapped onto this linear array of storage locations by the compiler. The compiler stores EACH ROW of a two-d array in successive memory locations.

Array In Memory

- Such a mapping of the elements of a two-dimensional array onto a one-dimensional array is called "ROW-MAJOR FORM“. The compiler uses the following address translation mechanism when accessing two dimensional arrays stored in row-major form in main memory two_d_array[i][j] = one_d_array[i*(number of columns) + j]
- The first row is placed at the start and then the next row begins immediately after the last value of the first row, placing them “tip to tail”.
- The offset within the row (column number) is added to the row number to determine total displacement along line to reach correct position.

Array In Memory

- For example, in the above example, if there are 4 columns in our two dimensional array, thus two_d_array[0][2] is found in one_d_array[0*4 + 2] = one_d_array[2] and two_d_array[1][3] is found in one_d_array[1*4 + 3] = one_d_array[7].
- This must include the actual size of the data stored, so this offset is multiplied by the size of the data in bytes to determine the actual displacement within memory.

Arrays in Memory

- It is important to note that the name of the array is associated with the starting address of the first element in the array. In reality, the compiler simply translates [row][column] indices of a two-d array into an offset from some starting address of where the elements of the array are stored in.

Assigning Values to arrays

- When assigning values within the declaration you may use the following syntax.
  - volt_recv[array_size]={1.76,2.11,1.19,1.99,1.22,3.01,1.88,1.75,1.88,1.49};
- In this case array size was a previously defined constant equal to 10 and the array is filled up one value separated by commas for each location.
- A two dimensional array can be defined in the declaration as follows (note array size is 3 in this case):
  - float array_sys[array_size][array_size]={{1,2,3},{2,5,3},{1,0,8}};

Assigning Values to arrays

- Within a loop the values are assigned one at a time:
  - for (int index=0;index<array_size;index++)
  - {
  - if (volt_read[index] > volt_max)
  - volt_warn[index]='O';
  - else if (volt_read[index]<volt_min)
  - volt_warn[index]='U';
  - else
  - volt_warn[index]='G';
  - }
- In this case a character array is filled up with values based on what is stored in another array with the same index value.

Examples of arrays

- On the website I have placed some examples of programs that use arrays.
- Program 21 computes a Fibonacci sequence with array operations rather than recursion.
- Program22 same thing but write output to a file.
- Program 23 reads a data array and based on the values within the array writes out results of test to another array using index to relate the two arrays.
- Program 26 performs Naïve Gaussian elimination on matrix and solves 3x3 system of equations.
- Program 27 same thing but output to a file.
- Program 28 reads a character file and prints out the equivalent ASCII code.

Memory Access out of bounds

- Array names act much like pointers in that they provide access to positions within memory.
- The C++ compiler that we use does not limit the access to the bounds of the array. This means that you can read and write from other locations within memory.
- This can lead to run time errors (problems that occur when the program is running) that are hard to detect. Unlike compile time errors (compiler will tell you there is a problem and stop) these errors can be hard to find and hard to fix.

Global Declarations

- It is also possible to declare arrays as global (for example having them declared before the main program and functions).
- This will allow them to be accessed by all functions.
- This can be useful in many situations but not a critical since functions will access the arrays directly when the function is part of the function call.
- Note that when the program runs it will use the data type of the array in the same way that it would a variable so trying to put a float value into an int array location will result in the int part of the float beign stored and the rest lost.

Context of Arrays

- Arrays provide an easy way to manage large data sets as well as an intuitive method to link different data sets in terms of an index.
- Any looping method will work well with arrays but due to the finite dimensions of arrays for loops are often used with multidimensional arrays.
- When we discuss file input and output arrays will again be used as they provide a convenient method to keep track of the data.

Next Lecture

- In the next lecture we will discuss pointers.