Arrays

What is it? An array is a data structure that holds a number of related variables. Thus, an array has a size which is the number of variables it can store. All of these variables must be of the same type. (In essence declaring an array allows you to use many variables of the same type without explicitly declaring all of them.) You can picture an array like below:

Each cell of the array can hold one data item. Furthermore, each cell has its own index, to distinguish it from the rest of the cells. In C, these cells are always numbered from 0 to the size of the array minus one.

Here is the generic syntax for an array declaration:

```
type  <var_name>[size];
```

For example to define an integer array called numbers of size 10, we would do the following:

```
int numbers[10];
```

Note that the expression inside the brackets of an array declaration must evaluate to a constant. This is because the computer needs to know how much space to allocate for the array beforehand.

To refer to a variable in a single cell or element of an array, you use the name of the array, followed by a bracket, the index you want to access, and then another bracket. Thus,

```
numbers[0] = 3;
```

would set the int variable in the 0 index of the array numbers to 3.

A sample program

The following program will read in five test scores and then print these out in reverse order:

```
int main() {

    int index, test_scores[5];

    // Read in scores into the array.
    printf("Please enter 5 test scores.\n");
    for (index=0; index < 5; index++)
        scanf("%d", &test_scores[index]);

     // Print them out by going through the array in backwards.
     printf("Here are the scores in reverse order: ");
     for (index=4; index >= 0; index--)
        printf("%d  ", test_scores[index]);

}
```

Common mistakes made with arrays

1. **Out of Bounds error: This is when you index into an array with an invalid index. For example, if you declare an array as follows:**

   ```
   int test_scores[5];
   ```

   and tried to access test_scores[5] or test_scores[-1], you would be trying to access a variable that does not exist, since the valid indeces of test_scores range from 0 through 4. Often times, these errors are masked by the fact that the index to an array in an algorithm can be a complex expression. It is difficult to tell whether a particular expression will equal an invalid index at some point during the execution of the algorithm.

2. **Not initializing all values of an array.**

3. **Trying to assign an entire array a value such as:**

   ```
   test_scores = 7;
   ```

   This will not assign 7 to each element of the array test-scores. In fact, it is not syntactically correct. This is because the left-hand side is not an integer variable, while the right hand side is an integer expression.

4. **Not differentiating between an array index and the value stored in an array at a particular index. (We will talk about this more as we see sample algorithms.)**

What’s relatively simple about arrays?

Once you understand the difference between an array, array element, and an index to an array, it is fairly simple to follow array code and to write syntactically correct code dealing with arrays.

What is difficult about arrays?

The actual manipulation of array elements and indeces can get quite tricky. Let’s look at an example, similar to the one we just did.

Let’s say you are given an array named numbers, indexed from 0 to 99. You are to reverse the contents of the array. It looks like this solution might work:

```
int index, temp, numbers[100];

for (index=0; index<100; index++) {
    temp = numbers[index];
    numbers[index] = numbers[99 – index];
    numbers[99 – index] = temp;
}
```

But, it does not. What happens here?

You end up reversing the list twice, so by the end of the loop, you’ve got the array in the exact order that you started.

Finally, we realize that we need to go through half of the list instead of the whole list, while we are swapping numbers. So, here an algorithm that correctly reverses the values in the array:

```
int index, temp, numbers[100];

for (index=0; index<100/2; index++) {
     temp = numbers[index];
     numbers[index] = numbers[99 – index];
     numbers[99 – index] = temp;
}
```

Searching for a value in an array

One of the most common tasks performed on an array is a search for a particular value. Let’s look at a straightforward way to do this in a segment of code. Assume the array numbers is already filled with values and that all of the variables used have already been defined appropriately.

```
printf("What is the number to search for?\n");
scanf("%d", val);

for (index=0; index < length; index++) {
   if (numbers[index] == val)
      found = 1;
}

if (found == 1)
   printf("%d was in the array.\n");
else
   printf("%d was NOT in the array.\n");
```

Character Counting Example

Consider the following task:

Given an input of characters, count the frequencies of each alphabetic character, and print all of these out.

First, I will write a program without functions that completes this task. Then, I will show an improved design of this program with functions, tracing through the differences in this second implementation.

Here are some of the issues we'll have to discuss:

1) How to create 26 different counters using an array.
2) How update the proper counter once a letter is read in.
3) How to print out all the values of the counters in a reasonable manner.

Here are quick answers to these questions:

1) Use an array of size 26, and initialize each element to 0.

2) First we can test to see if the character is a letter. If so, we must determine how to find its corresponding value from 0 to 25. (Hint: Use the idea given in the last lecture!)

3) We can loop through each index (0 to 25) into the array, and for each one, print out the corresponding letter and value stored in the array at that index.

```c
#include <stdio.h>

int main() {

  int index, freq[26];
  char ch;
  FILE* fin;

  fin = fopen("input.txt", "r");
  for (index = 0; index<26; index++)
    freq[index] = 0;

  while ((ch = fgetc(fin)) != EOF) {
    if (isalpha(ch))
      freq[tolower(ch)-'a']++;
  }

  printf("Letter\tFrequency\n");
  for (index = 0; index<26; index++) {

    printf("%c\t%d\n",(char)('a'+index),
                       freq[index]);
  }

  fclose(fin);
  return 0;
}
```