EE 1301
Lab 6: Arrays[:-]

Arrays

Many times in a program we may want many variables of the same type. For example, let's say I am simulating the behavior of a circuit that contains 1,000,000 gates, and I need to keep track of the output value (0 or 1) of each gate. Can you imagine the difficulty of declaring 1,000,000 separate variables in your program (or even 100, for that matter) and writing 1,000,000 lines of code every time you want to update the value of each variable? Luckily, we don't have to, because programming languages like C++ use an important concept called **arrays**.

An array is a series of elements of the same type placed in contiguous memory locations that can be individually referenced by adding an index to a unique identifier. The unique identifier is a name that points to the first element in the array, and we can get a reference to any other element in the array simply by adding an offset to that pointer. That means that, for example, 1,000,000 values of type int can be declared as an array without having to declare 1,000,000 different variables (each with its own name). Instead, using an array, the 1,000,000 int values are stored in contiguous memory locations, and all 1,000,000 can be accessed using the same identifier, along with the proper index or offset.

In our example above, instead of declaring variables `gate0` through `gate999999`, we can simply declare an array with 1,000,000 elements called `gates[]`. If I want to access the 100th gate, I simply write `gate[99]` (since array indices start at 0). I can access the first element with `gate[0]`. This is very powerful, because now I can make updates to all the gates in my array using a few lines of code. For example, I can initialize the value of every gate to 0 with the simple loop below.

```
for(int i=0; i<1000000; i++){
      gate[i] = 0;
}
```

Arrays or *lists* of data are integral to problem solving in every programming language. In this lab, we begin exploring this important concept.

Mystery-Box Challenge!

Here is your next mystery-box challenge. Determine what the following function does and explain it to your TA.

```
bool mystery(string fstr)
{
    string rstr;  // a string is like an array of chars, e.g., char[]
    for(int i=fstr.length()-1; i>=0 ;i--){
        rstr +=  fstr[i];  // fstr[i] gets the char at index i
    }
    return rstr == fstr;
}
```

Warm-up

1) **Declaring / initializing arrays**
Download partner.cpp from Canvas. Currently, there are two name variables (fullName1 and fullName2) and two height variables (height1 and height2). Rewrite the code to use an array for each pair of variables (one array for all names and another array for all heights). From the perspective of someone running the program, it should look exactly the same (i.e., same input and output).

2) **Passing arrays to functions**
Make a copy of your code from **warm-up 1** to a new file as a starting point for this problem. In `main(),` there is a `cout` statement. Make a new function that displays the same information as this `cout` and replace the code in `main()` with a call to the new function. **(Hint:** Cut and paste the `cout` statement into a function and pass the correct arguments.)

Stretch

1) **Partially-filled arrays**
Make a copy of your code from **warm-up 2** to a new file as a starting point for this problem. Modify your `main()` function to allow any number of names and heights to be entered (instead of just 2). You can assume that there will not be more than 100 people. (**Hint:** You can declare arrays that are big enough to hold all the names and heights in any case and only partially fill the arrays when fewer than 100 entries are needed.

Workout

1) **Concatenating arrays**
Strings have build-in functions to perform concatenation. Write a function called `append()` that performs concatenation for two char arrays (you will have to pass in a third char array that will hold the result). You will also need to pass in the lengths of each char array. You may not use strings for this part.

Sample main code:

```
char first[] = {'I', ' ', 'a', 'm', ' '};
char second[] = {'i', 'r', 'o', 'n', 'm', 'a', 'n', '\0'};
char result[200];
append(first, 5, second, 8, result);
cout << result;
```

Example output for code above:

```
I am ironman
```

2) **Bubble Sort**
Sorting a list of numbers is an important computing problem that has been extensively studied. One of the simplest methods is known as **bubble sort**. Given a list of numbers, bubble sort sorts the list by passing over the list multiple times and moving at least one element into the correct position in each pass. Therefore, a list of N elements is guaranteed to be sorted after at most N-1 passes. Consider the following list.

```
3, 5, 2, 8, 9, 1
```

The basic idea of bubble sort is to start at the beginning of the list and compare the first two numbers in the list and swap them if the first one is larger than the second (assuming you wish to sort from low to high). Next, the second and third numbers are compared and swapped if necessary, then the third and fourth, fourth and fifth, and so on until the entire list has been examined. After the **first pass** over the list, the **largest value will *always* end up in the last position,** as shown below.

```
3, 2, 5, 8, 1, 9
```

The name bubble sort refers to the fact that large numbers "bubble up" to the top of the list as the algorithm runs. We repeat the process, "bubbling" the larger values up in the list on each pass. Note, however, that for the second pass we don't need to examine the last value, because it's guaranteed to be the largest. After the second pass, the last *two* values need not be examined, and so on.

The process **ends** when an entire pass through the list results in **no values being swapped**.

Write a program that will implement the Bubble Sort algorithm. For this problem, you need to do the following:

- Declare an integer array named *list* that contains 50 values.
- Using a loop, initialize *list* with the values 100, 99, 98, … (in decreasing order)
- Construct a void function named bsort that implements the Bubble Sort algorithm, as described above. Your function should accept two arguments: the integer array to be sorted and the number of elements in the array.
- Call the bsort function to sort the *list* in increasing order.
- Print out the elements of *list*, 5 per line as follows:

Example:

```
...
51    52      53      54      55
56    57      58      59      60
...
```

[**Hint:** This problem will be much easier if you attack it in stages. One plan of attack is as follows. First, write a swap function that will swap two integer values. Then, write a function that will make a single pass through the array, calling the swap function to correct any out-of-order elements. Finally, call this second function as many times as necessary to sort the array.]

Debugging

Use gdb to debug your program. Note that you can use the print command to print an entire array, or just a portion of an array. Try inspecting the array contents in between iterations of bubble sorting to confirm that your algorithm is working as expected.

Check

List one important thing you learned in lab today, one question you still have about the lab material, and one way arrays can be used in electrical engineering. When you are done, share your list with a neighbor or discuss with the TA.

Challenge

Challengers, mount up!

1) **Non-repeating letters**

Write a `noRepeat()` function that takes three character arrays and two integers as arguments. The `noRepeat()` function should first concatenate two of the char arrays (using workout 1), then copy the letters into the third char array **without any repeated characters** (only the first time the character is encountered should it be copied to the third array). After copying into the third array, add a null terminator character (i.e., `'\0'`) at the end to ensure that `cout` does not print funny things. Sample `main()` code is provided below. Do not use strings for this challenge.

Sample main code:

```
char x[] = {'h', 'e', 'l', 'l', 'o', ' '};
char y[] = {'l', 'l', 'a', 'm', 'a'};
char result[200];
noRepeat(x, 6, y, 5, result);
cout << result;
noRepeat(y, 5, x, 6, result);
cout << result;
```

Example output for above:

```
helo am lamheo
```