Lecture 3 Notes

Arrays

An array is a list of values stored in one variable. Each value in this list is known as an element, and it can be accessed using an index.

An array declaration looks like this:

```
double a[100];
```

This array has 100 elements of type double. The elements in this array are indexed from 0 to 99. In general, an array containing N elements is indexed from 0 to $N - 1$. To access the first element of the array, use the expression `a[0]`. To access the 2nd element, use the expression `a[1]`. To access the last element of a 100-length array, use the expression `a[99]`.

Remember that each array element is like a variable where you can store or retrieve a value, so you may use the assignment operator on an array element (for example: `a[5] = 9.81`).

When retrieving an element by index, you may also use the value of an integer variable for the index. For example, you can loop through the uninitialized elements in an array and fill them with values input by the user:

```
int i;
double a[10];

cout << "Enter 10 numbers: ";
for (i = 0; i < 10; ++i) {
    cin >> a[i];
}
```

While uninitialized arrays need to be declared with a size, you do not need to specify a size if the array is a parameter to a function:

```
int sum(int a[], int len)
{
    int i, s;

    s = 0;
    for (i = 0; i < len; ++i) {
        s = s + a[i];
    }
    return s;
}
```

This function calculates the sum of an integer array. The first parameter is the array and the second parameter is the number of elements in that array. This function works on arrays of any length; however, if you needed a function that operated on an array of a specific length, you can specify a size inside the square brackets of the array parameter.

Character Arrays

A character array stores elements of type char. They are useful for storing textual messages displayable to humans, when used in this fashion, a character array is known as a string. We can store text data from user input into a character array like this:

Listing 1: name.cpp

```
#include <iostream>
using namespace std;

int main()
{
    char name[100];

    cout << "Enter your name: ";
    cin >> name;
    cout << "Hello, " << name << "!" << endl;
    return 0;
}
```

The character array in this example is limited to 100 characters, so the program will break if you type input that exceeds this limit. In the next lecture you will learn about std::string, which is supposed to be a better alternative to strings. Strong proponents of std::string will refer to them simply as “string”, while calling a string stored as a character array: “C-string”.

The statement `cin >> name` reads one “word” from standard input and stores it into name. A word is a sequence of non-whitespace characters. To read an entire line from standard input, we would use the statement: `cin.getline(name, 100)`. Where the integer 100 refers to the capacity of our array.

Strings stored as character arrays are often discouraged in C++ ever since the marvelous invention of std::string. Unlike other data types, you cannot use the assignment operator on a character array, nor can you use any of the comparison operators. Thus, statements like:

```
char s[32];
s = "This does not work";
```

Is invalid. Arrays are simply used for storage, they cannot be operated on like scalar variables. To assign new content to an array requires that you set each desired element to a new value. To compare if two arrays are “equal”, requires that you compare the elements of the two arrays one at a time.

Fortunately, you can include the header cstring to use functions that operate on old-fashioned strings. To assign a string into a character array, use strcpy:

```
char s[32];
strcpy(s, "This works");
```

Remember that you must use `#include <cstring>` to use strcpy.

As was said, you cannot compare two strings using the equality operator. So this will not work:

```
char s[32];
strcpy(s, "Apples");
cout << (s == "Oranges");    /* <--- Wrong */
```

This is the correct way to compare strings:

```
char s[32];
strcpy(s, "Apples");
cout << (strcmp(s, "Oranges") == 0);
```

The function strcmp returns zero when both string arguments have the same content. It returns a value less than zero when the first string is alphabetically lesser than the second string. When the first string is alphabetically greater than the second string, strcmp returns a value greater than zero.

Two other useful string functions you should know about are strcat and strlen. I will not give examples of using these, it is important to learn how to use a function given a description and a "prototype" (declaration of the function without it's inner code) of the function:

```
/* Concatenates string b to string a
 * Returns string a.
 */
void strcat(char a[], const char b[]);

/* Returns the length of a string */
int strlen(const char s[]);
```

These are not the exact definitions of strcat and strlen, but they work as expected if you use them in this way.

Pointers