CS 122 Lab 2

`Introduction to Arrays` Paul Cao

Arrays are one of the most important and useful data types in programming languages. In this lab, we will review how to define, initialize, and access arrays and array elements. More advanced topics on arrays will be addressed in the next lab. Please turn in your report for the in-lab section before midnight on Thursday, 1/27/2011.

Pre-lab

Read the following reviews. You don't have to turn in anything for the pre-lab.

So far all of our variables have been able to hold only one value at any one point in time. Such variables are called scalar variables. Now it is time for our first non-scalar variable, an array.

An array is a variable capable of storing multiple values. When we declare an array we tell the compiler how many values we want the array to hold. We also tell the compiler what type of values the array can store. All of the values in an array must be of the same type.

Here is a declaration of an array called `numlist` that will be used to store 5 integers:

```
int numlist[5]; // declaring an integer array that can store 5 values
```

Each of the integers in the array is stored in the same number of bytes as a scalar integer, which on most machines is 4 bytes. Thus, the entire array will occupy 20 bytes of memory. The compiler always stores an array in contiguous memory locations (all of the elements of the array are stored in one chunk of memory with no gaps). Here is one way you may visualize the above array `numlist` when it stores the following 8 integers: 12, 8, 10, 123, 1000. Suppose that the compiler allocates the start of the array at memory address 1000.

Memory Address	1000	1004	1008	1012	1016
Array Index	0	1	2	3	4
Indexed Variable	`numlist[0]`	`numlist[1]`	`numlist[2]`	`numlist[3]`	`numlist[4]`
Array Content	12	8	10	123	1000

The individual values stored in an array are called the elements of the array. You will also hear them called indexed variables or subscripted variables. Each of the elements of an array is assigned an index. An index is a natural number in the range {0,1,2,...}. **Note that the array index always starts from 0 in almost all programming languages.**

As it is shown in the above table, to access one of the elements of an array, you put the index of that element in square brackets after the name of the array. The 0th element in the array called `numlist` is `numlist[0]`, the next one is `numlist[1]`, and so forth. Since we start the numbering with 0, the last element in `numlist` is `numlist[4]`.

To put the value of 12 into the 0th element of `numlist`, we will use:

```
numlist[0] = 12;
```

If we wanted to store a value that is entered from the keyboard into element `numlist[1]` we use:

```
cin >> numlist[1];
```

An array element like `numlist[4]` can be used in any way that a scalar variable can be used. All of the following statements are legal:

```
if (numlist[2] > numlist[1])  // Compares the third element with the second element
cout << numlist[3];           // Displays the fourth element of the array
sum = sum + numlist[0];       // Adds the first element to sum
```

The index inside the square brackets does not have to be an integer constant such as 1 or 2. It can be any integral expression that evaluates to an integer within the permitted range of the array's index. So an expression such as this:

```
for (i = 0; i < 2; i++)
      numlist[2*i+1] = 0;     // set the odd elements of the array to 0
```

If you wish to fill the array `numlist` with the integers typed from the keyboard, you can use a `for` loop too. It might be easier for our user to keep up with different values that need to be entered if we display something more helpful than `"Enter the next value: "`. Since users typically number items in a list starting from 1, we will say `"Enter value #1: "` when asking for `numlist[0]`, `"Enter value #2: "` when asking for `numlist[1]`, and so forth. By asking for value 1, then value 2, etc., we are allowing our user to count in a more natural way than C++ forces us to count. That is the most confusing part of working with arrays. It is natural to think that an array of size 5 will keep 5 values, thus, assuming that the indices would be 1 through 5. For an array of size 5, index 1 is a valid index, but index 5 is invalid and will cause a run-time error if it is used.

The following program allows you to enter 5 integers from the keyboard and will store those values in array `numlist`.

```
// lab2.cpp - A program that uses an array of integers
#include <iostream>
using namespace std;

int main (){
    int numlist[5], i;

    // Read 5 integers from the keyboard
    for (i = 0; i<5; i++ ){
      cout << "Enter value #" << i+1 << ": ";
      cin >> numlist[i];
    }
    // Display the numbers in a reverse order
    for (i = 5 ; i > 0 ; i-- ){
      cout << "Value #" << i << ": ";
      cout << numlist[i-1] << endl;  //Pay attention to i-1!
    }
    return 0;
}
```

Array Index Out of Range
A common error when working with an array is attempting to access an element of the array using an index that is out of range. In the above program, array `numlist` has 5 elements. The final value is called numlist[4]. If we try to access `numlist[5]`, some C++ compilers will not give us an error message at run time. C++ does not verify that your index is within the proper range when you compile the program. If you print the value of `numlist[5]`, the compiler will grab the 4 bytes following `numlist[4]`, interpret whatever is stored there as an integer and print the value of that integer. If you store a value at `numlist[5]`, the compiler will place that value into the first 4 bytes following `numlist[4]`, even if those bytes happen to be storing a different variable!

Initializing Arrays
A scalar variable can be initialized when it is declared, like this:

```
int num = 4;
```

An array can also be initialized when it is declared. Here we put the value 12 into `numlist[0]`, the value 8 into `numlist[1]`, etc.:

```
int numlist[5] = {12,  8, 10, 123, 1000};
```

If you list fewer values within the braces { and } than the declared size (5 in the above example) of the array, our C++ compiler will initialize all the rest of the elements to 0. However, not all C++ compilers will do this. If you initialize an array when it is declared, you can omit the size of the array. C++ will use the number of initializers in your list as the size of the array. Here is an example:

```
char vowels[] = {'a', 'e', 'i', 'o', 'u'};
```

This declared a character array of size 5 which stores the lowercase vowels, a, e, i, o, and u.

Creating Arrays of Flexible Size
One way to create an array with a particular size is to use a global variable to define the size of that array. Thus, every time one can change that number to increase or decrease the size of an array. This requires you to recompile the program for the new size to take effect. Let's modify program lab2.cpp to work on flexible size arrays.

```
// lab2b.cpp - A program that uses a flexible size array of integers
#include <iostream>
using namespace std;
const int SIZE = 5;  // Set the maximum size for the array

int main(void){
    int numlist[SIZE];

    // Read SIZE integers from the keyboard
    for (int i = 0; i<SIZE; i++ ){
      cout << "Enter value #" << i+1 << ": ";
      cin >> numlist[i];
    }
    // Display the numbers in a reverse order
    for (int i = SIZE; i > 0; i-- )
    {
      cout << "Value #" << i << ": ";
      cout << numlist[i-1] << endl;  //Pay attention to i-1!
    }
    return 0;
}
```

This produces the same result as lab2.cpp. Now, you are limited to array of 5 integers. By changing the value for `SIZE`, you can read as many numbers as you wish.

In-Lab Activity

Use one-dimensional arrays to solve the following problem.

Because of the economic situation, most car dealers are raising the commission of car salesperson. A year ago, a commission of 25% of the profit is common but now 35% has become a standard practice, especially for American car dealers. For example, if the dealer bought a car of $20,000 and the salesperson sold it for $24,000, then the salesperson will get a commission of (24000-20000)*35%=1400.

Develop a C++ program (using two array of doubles) that will first read in the sales data and calculate the commission of the salesperson. Your program will prompt the user to give you two values, for each transaction, the base price of the car and the sale price of the car. Suppose a salesperson at a Chrysler dealer sold 6 cars this month (that's a decent record in the car business). After all input data have been given your code outputs the commission for each car. In the end, you should also display the total commission of the salesperson earned in the month. You can assume that a salesperson won't be able to sell more than 20 cars in a month and he/she will sell at least one car in the month. This means that the arrays in your code may be partially. The user will indicate the end of his/her input by entering END for the car model (all capital letters in END). You can use the following two data sets to test your code.

Data set 1

Model	**Base Price**	**Sale Price**
2007_Pacifica	12102	14755
2009_300	38903	39290
2008_Pacifica	23000	29890
2008_PT_Cruiser	18250	20520
2008_Sebring	21000	21255
2008_Town_&_Country	35480	38250

Data set 2

Model	**Base Price**	**Sale Price**
1991_Pontiac_Firebird	2466	2979
2007_Ford_F_150	11296	13900
2009_Chevy_Cobalt	10002	10995

Note

1. You should have four arrays in your program and all of them may be partially filled. One array is used to store the base price, one for the sale price, and one for the commission. They should be of type `double`. The fourth array should be `string` type and is used to store the model.

2. When you input models into strings, don't have spaces in the model or else the model name can't be fully transferred into your variable. For example, when you input model for "2007 Chrysler Pacifica", type "2007_Chrysler_Pacifica" so there is no white spaces between the words. We will learn more about string operations in chapter 8.
3. Don't forget `#include <string>` directive in the front of your code.

Sample output with dataset 1:

```
Welcome to our commission calculation system!

Car #1
Please input the car model (use _ to replace space in the model name): 2007_Pacifica
Please input the base price: $12102
Please input the sale price: $14755

Car #2
Please input the car model (use _ to replace space in the model name): 2009_300
Please input the base price: $38903
Please input the sale price: $39290

Car #3
Please input the car model (use _ to replace space in the model name): 2008_Pacifica
Please input the base price: $23000
Please input the sale price: $29890

Car #4
Please input the car model (use _ to replace space in the model name): 2008_PT_Cruiser
Please input the base price: $18250
Please input the sale price: $20520

Car #5
Please input the car model (use _ to replace space in the model name): 2008_Sebring
Please input the base price: $21000
Please input the sale price: $21255

Car #6
Please input the car model (use _ to replace space in the model name): 2008_Town_&_Country
Please input the base price: $35480
Please input the sale price: $38250

Car #7
Please input the car model (use _ to replace space in the model name): END
The commissions are the following.
2007_Pacifica         $928.55
2009_300                  $135.45
2008_Pacifica         $2411.50
2008_PT_Cruiser       $794.50
2008_Sebring          $89.25
2008_Town_&_Country   $969.50
```

Sample output with dataset 2:

```
Welcome to our commission calculation system!

Car #1
Please input the car model (use _ to replace space in the model name): 1991_Pontiac_Firebird
Please input the base price: $2466
Please input the sale price: $2979

Car #2
Please input the car model (use _ to replace space in the model name): 2007_Ford_F_150
Please input the base price: $11296
Please input the sale price: $13900
```

```
Car #3
Please input the car model (use _ to replace space in the model name): 2009_Chevy_Cobalt
Please input the base price: $10002
Please input the sale price: $10995

Car #4
Please input the car model (use _ to replace space in the model name): END
The commissions are the following.
1991_Pontiac_Firebird $179.55
2007_Ford_F_150       $911.40
2009_Chevy_Cobalt     $347.55


The total commission of this month is $1438.50
```