Lab 10 - CMPS 1043- Computer Science I – Arrays

Objectives: Demonstrate use of statically allocated arrays in a variety of scenarios:
1. Initializing an array 2. Searching an array 3. Counting values in an array

An array in C++ is a named collection of values of a specific type stored in a sequence. Although they may contain integers, floats, booleans, doubles or even structures, each array contains only a single type. Here are some example uninitialized declarations.

Declarations: These declare space but do not contain any values as of yet.

- `int A[10]  // Allocates an array that can contain 10 integers.`
- `float ave[100]  // Allocates an array that can contain 100 floats.`

Note: You cannot initialize an array size with a *variable*. I.E. `int X[size]`. Later you will be shown a way to do this using dynamic memory allocation. You can however initialize the size of an array with a constant.

- `const int SIZE = 20;`
- `int scores[SIZE];  // This works since constant values are handled`
  `// during compile time.`

Accessing the contents of an array – Individual values in an array are accessed using *subscripting*. The first value in the A[10] array can be accessed via A[0], the second value by A[1] and so on up to A[9]. **NOTE**: that the subscripting starts at 0 and goes up to 9, NOT 10. A variable can also be used for the subscript, as long as the variable is in the range of the legal subscripts. You can utilize a subscripted variable in the same ways you any other variable.

Examples:

```
cin >> A[1] >> A[2*3];
cout << A[5] << A[2 + X];
if (ave[99] >= 50]…
while (ave[4] <= A[y])….
Z = A[4] + 10;
```

The contents of an array can be initialized in a variety of ways. It you do not initialize the array, each location contains garbage. The following is a simple way to initialize every slot to the same value.

- `int Values[20]={0};  // Initializes every array location to zero.`

If you have different values you want to place in each slot you can do something like this.

- `char symbol[3]={'x', 'y', 'z'};`
- `int count[5]={2,4, 6, 8, 10, 12};`
- `int count[] ={2,4,6,8,10,12}; // equivalent to the previous line.`

Of course you can always do something like the following.

```
count[0]=2; count[1]=4; count[2]=6; count[3]=8;
count[4]=10; count[5]=12;
```

Or by using a loop and a little math

```
for (int i=0;i<5;i++) count[i]= 2*(i+1);
```

What values are in the array after the for loop?

Using a while to initialize an array – Write a program that includes the following code then execute. This will allocate an array named Values with the numbers from 1 to 10. It will then print out these values.

```
int sub = 0; // This is our subscript variable. Starts out at 0
int Values[10]; // Here is the array declaration
// Load the array
while (sub < 10) // NOTE: This is not less than or equal.  Why?
{
  Values[sub]= sub+1; // Note that when sub=0 this assignment is
                      //  Values[0]=1; and so on.
}
// Print the contents of the array. We use a for loop here just to
// be different.
For(int i=0;i<10;i++){
  { cout << Values[i] << " "; }
cout << " END OF OUTPUT"<<endl;
```

Comments: Normally one uses a ***for*** loop to initialize, process and print arrays of KNOWN size. For example the while loop in the above case would be a ***for*** as well.

Initializing an array with random numbers – To make things more interesting let's allocate an array and initialize it to random throws of a die. In other words for each slot in the array we will randomly generate a number from 1 thru 6 and put that number in the array.

`rand()`: To generate numbers the ***rand()*** function will be called. (For more information, see pages 128 – 130 in your text.) This function returns a random integer from 0 to RAND_MAX. RAND_MAX is a constant defined in <cstdlib>. Its default value may vary between implementations but it is guaranteed to be at least 32767. The usual way to generate pseudo-random number in a specific range is to use the **modulo operator** (%). Here are some examples.

```
rand() % 5        // returns a number from 0 thru 4
rand() % 10 +1  // returns a number from 1 thru 10
```

Before using this function, the program should set the seed for the random number generator. This seed determines the sequence of random numbers that are generated. The command srand(int) is used to set the seed. A program that sets the same seed on each run will generate the exact **SAME** sequence of random numbers, a feature that is quite useful when you are debugging your programs. The following command can be used to generate a different sequence each time the program is executed. Of course you should not do this until after the program is debugged. These functions need the inclusion of <cstdlib> at the top of your program.

```
srand(time(NULL));// the values returned by calls to rand() will be different on every run of the program.
```

Type in the following program and run it. Don't forget necessary include commands.

```
srand(7);  // initialize a random number seed to 7
const int size=20;  // Note: this is a constant NOT a variable.
int Values[size]; // Here is the array declaration
// Load the array
for(int i=0;i<size;i++)
     {Values[i]=rand()%6+1;} // Loads a die throw.
```

```
// Print the contents of the array.
for(int i=0;i<size;i++)
   {cout << Values[i] << " "};
cout << "\nEND OF OUTPUT"<<endl;
```

Every time you run the above program you will get the same sequence. Make several runs and check it. Now change the seed and see what happens.

Counting the die values in the array (method 1) – Now, let's process the above array and count the number of 1's, 2's, ... 6's and see how close the values are to being "random". Normally, we should have about the same number of each.

Starting with the above program, remove the output command and add the following to the end. This will count only the ones and twos. Add the necessary code for the rest.

```
int one=0;two=0;//etc
for(int i=0;i<size;i++)
{ if(Value[i]==1) one++;
  if(Value[i]==2) two++;
    . . . .
}
// Write code to print out the contents of one, two and so on.
```

Several comments about the above program.

First, Could we have written the counting loop more efficiently? How?
Also, if we were throwing a twenty sided Dungeons and Dragons die (icosahedrons) would we need twenty variables. Hmmm…..

Counting the die values in the array (method 2) – In this section we will simplify the data collection by using another array. Continuing with the die problem we will allocate an integer array Count of size 7 to contain the different die counts. In other words, Count[1] will hold the number of 1's, Count[2] the number of 2's and so on. Since the contents of Value[i] is always 1 through 6 we can use it to subscript Count. The following is an example of this process.

```
int  Count[7]={0};// initialize it to zero.
for(int i=0;i<size;i++
{   Count[Value[i]]++;// Increment the appropriate Count .
}
// Write code to print out the contents of Count properly.
```

LAB ASSIGNMENT

Write a program that will load an array of size 100 with random numbers as generated by rand(). These will be in the range from 0 to 32K. Use srand(5) to set the seed. Add to this code a section that will search the array and print out the largest number found and its position. Also calculate and print the average of all the values.