

# LABORATORY 11

---

## Hurray for arrays

### Objective

This week in lab you will develop skills in manipulating arrays in C++. An array is a collection of objects of the same type that are referenced through a common name. One of the applications you will consider is searching for a particular value in a list of values.

### Key Concepts

- Single-dimension arrays
- Subscripting
- Array manipulation
- Improper indexing pitfalls
- Passing arrays
- Linear search
- Binary search

## 11.1 GETTING STARTED

- Using the procedures in the introductory laboratory handout, create the working directory `\cpplab` on the appropriate disk drive and obtain a copy of self-extracting archive `lab11.exe`. The copy should be placed in the `cpplab` directory. Execute the copy to extract the files necessary for this laboratory.
- Many of the activities that are performed in the laboratory can be done in groups but you should work the exercises yourself.

## 11.2 ARRAY BASICS

- Open the program file `five.cpp`. This program first extracts five values from a standard input stream to set the elements of an array. The program then displays the values of the array. Although the program is correct for its task, the actual implementation can be improved.
- Modify the program so that it extracts and lists eight numbers. Run your program after it has been modified. (You will need to create a default workspace).
- Were you required to make an additional modification to get the program to work? Should this modification have been necessary or should the original designer/implementer have written the program differently? Is there an appropriate motto for dealing with arrays and constants? ✓
- Now modify the program so that it displays the minimum value in the list before displaying the values in the entire list. The following screen capture demonstrates the desired input/output behavior.



```
(Inactive C:\WINDOWS\DESKTOP\EIGHT.EXE)
Enter a list of 8 elements
Number: 32
Number: 8
Number: 16
Number: -4
Number: 64
Number: 128
Number: 2
Number: 4

The minimum entered value is -4

The entered list is:
Number[ 0]: 32
Number[ 1]: 8
Number[ 2]: 16
Number[ 3]: -4
Number[ 4]: 64
Number[ 5]: 128
Number[ 6]: 2
Number[ 7]: 4
```



- Demonstrate that your modified program works correctly. ✓
- Modify the program again so that when displaying the values in the list, if the current element being processed is equal to the minimum value, the



string "minimum" is displayed at the end of the output line. Demonstrate that your program works correctly. ✓

- Close the current workspace.

One of the most common errors when using arrays is trying to use an invalid array index. C++ will not stop you from overrunning the end of an array. Therefore, the programmer must ensure that the array is always defined large enough to hold the necessary information and never allows an invalid index to be used.

- Open the program file `sum.cpp`. This program initializes two arrays A and B and produces a third array C whose values are the sum of A and B.
- Write down the problems you see with the program in the space provided below.

- In C++, when using a `for` loop to iterate through an array you usually want the loop indices to range from 0 to  $n-1$ , where  $n$  is the size of the array. This design corresponds to the fact that C++ numbers array indices from 0 to  $n-1$ . Hence, the typical `for` loop initializes its index object to zero and repeats the loop as long as the index is less than the size of the list.
- Correct the errors in the program.
- Step through the program and watch the values of `A[0]`, `B[0]`, and `C[0]`. (You will need to create a default workspace). What happens as the program is running through the `for` loops? These values are not the values you want to watch. You really want to watch the current values being considered, which you can do by editing the watches and replacing the 0's with `i`. Step through the program again, but this time watch `A[i]`, `B[i]`, and `C[i]`. It is also possible to watch an entire array at one time. Add watches of A, B, and C and again step through the program. ✓



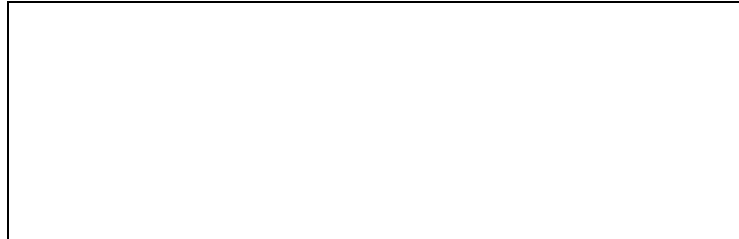
- Close the current workspace.

## 11.3 RE-DEJA VU

- Examine the following function and determine why it will not compile.

```
void f(const int A[]) {
    for (int i = 0; i < 10; ++i) {
        if (A[i] == 0) {
            A[i] = 1;
        }
    }
}
```

```
void f(const int A[]) {  
    }  
}
```




- Examine the following function and determine why it will not compile.

```
void g(const char A[]) {  
    cin >> A;  
}
```



- Examine the following function and determine why it will not compile.

```
void h(int &A[], int &n) {  
    int InputValue;  
    for (int n = 0; cin >> InputValue; ++n) {  
        A[n] = InputValue;  
    }  
}
```



- Examine the following program and determine its output.

```
#include <iostream>
#include <iomanip>
using namespace std;
const int MaxListSize = 100;
void f(int A[], int n) {
    for (int i = 0; i < n; ++i) {
        A[i] = -A[i];
    }
    n = 5;
}
int main() {
    int MyList[MaxListSize] =
        { 2, -3, 5, 7, 11, -13, 17, -19, -23, 29};
    int ListSize = 10;
    cout << "Initial values: ";
    for (int i = 0; i < ListSize; ++i) {
        cout << setw(4) << MyList[i];
    }
    cout << '\n' << endl;
    f(MyList, ListSize);
    cout << " New values: ";
    for (int j = 0; j < ListSize; ++j) {
        cout << setw(4) << MyList[j];
    }
    cout << '\n' << endl;
    return 0;
}
```



- Open the program file dejavu.cpp.
- Run the program and compare your predicted results to the actual results. If there are differences, determine why. (You will need to create a default workspace).



- Show your results to a lab instructor. ✓
- Close the current workspace.

## 11.4 REVERSING

Your next task is to implement a function `Reverse()` that reverses the list of values represented by its parameters `S` and `n`. In particular, the elements of `S` are to be **char** values. The number of elements to be manipulated is the **int** value `n`. Note `S` is not a `string` object.

The easiest way of reversing a list of elements is to use a loop that for each iteration swaps another pair of elements—one element from the left half of the list and another element from the right half of the list. It is traditional to use two indexes `left` and `right` whose values are the indices into the two sublists. The loop iterates while the value of `left` is less than the value of `right`.

- Open the program file `reverse.cpp` and create a default workspace.
- Examine the function `main()`. It extracts a list, displays the list, reverses the list, displays the reverse list, restores the list using another reversal, and displays the now restored list. The list extraction and display are performed using functions that are also defined in `reverse.cpp`.
- Add the definition of `Reverse()` to `reverse.cpp`. Your implementation should use a function `Swap()` that you also define.
- Demonstrate that your program works correctly. ✓
- Close the current workspace.



## 11.5 SEARCHING


It is often important to search a list of data to determine if a value is present. You will now examine two methods of searching for a particular value: *exhaustive* and *binary*. You will also record how many element comparisons are necessary to complete each search.

### 11.5.1 Exhaustive searching

- Open the program file `exhaust.cpp`. This program will first read in the file `search.dat`. It will then prompt the user for a value. This value, called the *key*, is then compared to each value in the stored list. The program will display a message indicating whether the key is one of the list values.
- Run the program on the five suggested key values to complete the first column of the following table. Also compute the average number of values searched. (You will need to create a default workspace).

If we have randomly arranged data and a random key value, on average we will examine half the list to find the key value. If we know instead that the data is already ordered (sorted), then we can on average do much better.


| Search value | Search Technique              |                             |                     |        |                 |
|--------------|-------------------------------|-----------------------------|---------------------|--------|-----------------|
|              | Exhaustive with unsorted data | Exhaustive with sorted data | Modified Exhaustive | Binary | Modified Binary |
| 559          |                               |                             |                     |        |                 |
| 4442         |                               |                             |                     |        |                 |
| 2415         |                               |                             |                     |        |                 |
| 297          |                               |                             |                     |        |                 |
| 1173         |                               |                             |                     |        |                 |
| Average      |                               |                             |                     |        |                 |

- Modify the program to use the data file `sorted.dat`. This file is the same as `search.dat`, but the numbers have now been sorted.
  - Run the program and complete the second column of the table.
  - How has sorting affected the average? Probably not very much, since the program has not been modified. The difference occurs because of the data being rearranged. Let's now modify the program.
  - Because the data is sorted, we can stop searching for the key value once we find a list value that is greater than or equal to the one for which we are searching. Modify the program to perform the search this way. Your modification should cause the loop to be exited once the key value is less than or equal to the current item in the list that is being compared. The check after the loop may need to be changed depending on how you modified the loop. If necessary, change it appropriately.
- 
- Run the program again and complete the third column of the table. ✓
  - Close the current workspace.

## 11.5.2 Binary searching

The binary search described here is a more efficient search than the modified exhaustive search. For example, when you search for a name in a phone book, you probably don't start at the beginning and search straight through. You use the fact that the names are in sorted order to speed up your search.

The binary search is an iterative technique in which each iteration discards half of the remaining values from further consideration. The search starts with the middle list element and decides which half of the data to further examine. In the next iteration, the middle element from the half of the data values that can possibly contain the key value is computed, and from it the range of possible positions for the key value is further restricted. The process is repeated until the search finds the key value or until the search determines there are no more potential positions to consider.

- Open the file `binary.cpp`.
- Examine the program to determine how the binary search algorithm is implemented. In particular, notice that object `p` is used as an index of the middle element of the values currently being considered.
- Run the program and complete the fourth column of the table. (You will need to create a default workspace).
-  ▪ Is this search better or worse than the exhaustive approach? ✓
- Close the current workspace.

There is also an alternative to choosing a value for `p` that is sometimes more effective than always picking the middle element. The alternative supposes that the values in the list being processed are uniformly distributed over the interval determined by the leftmost and rightmost elements currently being considered. We can choose as the value for `p` the index of the element that is most likely to contain the key value.

Suppose a thousand values are in the list, and the value range is 1 ... 1,000. If the key value is 4, the expected location is the fourth element. Similarly, if the list contains 1,000 elements, the value range is 1 ... 2,000, and the key value is 1,500, the expected location is the 750th element in the list.

In general, if the current range of values being considered is  $a \dots b$  and if  $n$  is the number of values in that range, then the expected position  $p$  of the key value is

$$p = a + (n - 1) \frac{\text{key} - a}{b - a}.$$

- To modify a binary search to use the above formula, take the following into consideration:
  - For the “middle” position  $p$  to make sense,  $\text{key}$  must lie within the inclusive interval  $a \dots b$ . If  $\text{key}$  does not, it cannot be in the list. This



restriction requires two comparisons to be made in each iteration of the search loop to ensure that *key* is in the interval.

```
int a = List[left];
int b = List[right];
if ((key < a) || (key > b)) {
    comparisons += 2;
    break;
}
else {
    comparisons += 2;
}
```

- The formula for  $p$  cannot be evaluated if  $a$  and  $b$  are the same. Why?

This restriction means that another comparison needs to be made in each iteration of the search loop to determine whether  $a$  and  $b$  are the same. If they are different, the value of  $p$  is determined using the preceding formula.

```
if (a == b) {
    ++comparisons;
    spot = left;
    break;
}
else {
    ++comparisons;
}
int n = right - left + 1;
p = left + int((n-1) * float(key-a)/float(b-a));
```

In the preceding code, because the value of *key* has already been previously determined to be in the interval  $a \dots b$ , if the if-test indicates that  $a$  and  $b$  are the same, then  $a$  and  $b$  must both equal *key*. Why?

- Open the program file `modified.cpp` and complete the fifth column of the table. (You will need to create a default workspace). Is this search an improvement over the standard binary search? Discuss your results with a laboratory instructor ✓



## **11.6** FINISHING UP

- Copy any files you wish to keep to your own drive.
- Delete the directory `\cpp1ab`.
- Hand in your check-off sheet.