STUDENT OUTLINE

Lesson 17 – Single Dimension Arrays

A TYPE OF DATA STRUCTURE

INTRODUCTION: Programs often need a way to work with large amounts of data without declaring thousands of scalar variables. In this lesson we explain the terms *data structure* and *algorithm*, and then introduce you to a very important data structure, the *array*. With each new data structure comes the complementary algorithms to manipulate the data it stores. This combination of data structures and algorithms provide the powerful tools needed to solve computer science problems.

The key topics for this lesson are:

A. Data Structures and Algorithms
B. Example of an Array
C. Array Declarations and Memory Allocation
D. Applications of Arrays
E. Arrays as Parameters
F. Arrays and Algorithms

VOCABULARY:

DATA STRUCTURE
TRAVERSAL
SEQUENTIAL
INDEX
ALGORITHM
ARRAY
RANDOM ACCESS
`final`

DISCUSSION: A. Data Structures and Algorithms

A.K.A A PRIMITIVE

1. A scalar type is a simple data type that holds one value at a time. The data types `int`, `double`, `char`, and `boolean` are important and useful, but limited.

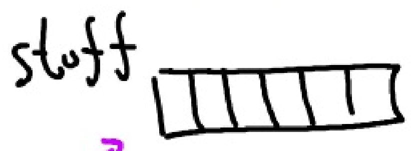
2. A data structure is a collection of scalar data types that are all referenced or accessed through one identifier.

3. Data structures can be created to store information about any real-world situation:
   a. Your high school transcript is a collection of grades.
   b. The English paper you wrote on a word processor is stored as a collection of characters, punctuation, and formatting codes.
   c. Your name, address, and phone number can be stored as a collection of strings.

4. After defining a data structure, algorithms will be needed to solve the specific problems associated with such a data structure.

5. One example of a data structure is an array. An array will store a list of values.

6. An algorithm is a sequence of programmed steps that solves a specific problem.



A CONTIGUOUS GROUP OF PRIMITIVES

VISIT EACH LOCATION IN THE ARRAY

7. The fundamental algorithms that apply to an array data structure are: **insertion, deletion, traversal, searching, and sorting.**

B. Example of an Array

1. The following program will introduce you to some of the syntax and usage of the *array* class in Java:

Program 17-1

DECLARE & INSTANTIATE A

TRAVERSE THE ARRAY

```
public class ArrayExample
{
  public static void main (String[] args)
  {
    int[] A = new int[6];  //  an array of 6 integers
    int loop;

    for (loop = 0; loop < 6; loop++)
      A[loop] = loop * loop;
    System.out.println("The contents of array A are:");
    System.out.println();
    for (loop = 0; loop < 6; loop++)
      System.out.print("  " + A[loop]);
    System.out.println();
  }
}
```

} YIELDS THIS

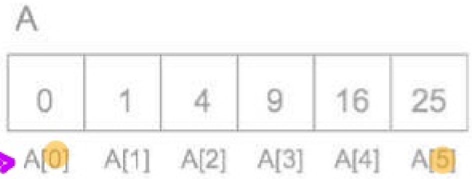
Run output:

```
The contents of array A are:

  0  1  4  9  16  25
```

(CONTIGUOUS)

2. An array is a linear data structure composed of adjacent memory locations, or “cells”, each holding values of the same type.



VERBALLY, "A SUB ZERO"

3. The variable A is an array, a group of 6 related scalar values. There are six locations in this array referenced by index positions 0 to 5. Note that indexes always start at zero, and count up by one's until the last slot of the array. If there are N slots in an array, the indexes will be 0 through N-1.

4. The variable `loop` is used in a **for** loop to reference index positions 0 through 5. In this program the square of each index position is stored in the memory location occupied by each cell of the array. The syntax for accessing a memory location of an array requires the use of square brackets [].

5. The square brackets `[]` are collectively an operator in Java. They are similar to the parentheses as they have the highest level of precedence compared to all other operators.

PROGRAM CRASH

6. The index operator performs automatic bounds checking. Bounds checking makes sure that the index is within the range for the array being referenced. Whenever a reference to an array element is made, the index must be greater than or equal to zero and less than the size of the array. If the index is not valid, the exception `ArrayIndexOutOfBoundsException` is thrown. Exceptions will be discussed in greater detail later, but they can immediately terminate program execution.

C. Array Declarations and Memory Allocation

1. Array declarations look like this:

DECLARE

```
type[] arrayName;
```

This tells the compiler that `arrayName` will be used as the name of an array containing ***type***. However, the actual array is not constructed by this declaration. Often an array is declared and constructed in one statement like this:

INSTANTIATE

```
type[] arrayName = new type[length];
```

This tells the compiler that `arrayName` will be used as the name of an array containing ***type***, and constructs an array object containing `length` number of slots.

2. An array is an object, and like any other object in Java is constructed out of main storage as the program is running. The array constructor uses different syntax than most object constructors; ***type***`[length]` names the type of data in each slot and the number of slots. For example:

```
int[] list = new int[6];
double[] data = new double[1000];
Student[] school = new Student[1250];
```

THIS IS WAY WE CALL THIS A STATIC DATA STRUCTURE

Once an array has been constructed, the number of slots it has does not change.

3. The size of an array can be defined by using a **`final`** value, which means that you cannot change it or derive from it later.

WISE

```
final int MAX = 200;
int[] numb = new int[MAX];
```

4. When an array is declared, enough memory is allocated to set up the full size of the array. For example, the array of `int` as described above,

```
int[] list = new int[6]
```

will require 24 bytes of memory (4 bytes per cell).

D. Application of Arrays

1. Suppose we have a text file *votes.txt* of integer data containing all the votes cast in an election. This election happened to have three candidates and the values in the integer file are 1, 2, or 3, each corresponding to one of the three candidates.

Program 17-2

```
import chn.util.*;

public class Votes
{
  public static void main (String[] args)
  {
    FileInput inFile = new FileInput("votes.txt");

    int vote, total = 0, loop;

    // sized to 4 boxes, initialized to 0's
    int[] data = new int[4];

    vote = inFile.readInt();
    while (inFile.hasMoreTokens())
    {
      data[vote]++;
      total++;
      vote = inFile.readInt();
    }
    System.out.println("Total # of votes = " + total);
    for (loop = 1; loop <= 3; loop++)
      System.out.println("Votes for #" + loop +
                         " = " + data[loop]);

  }
}
```

data IS BEING USED AS A FREQUENCY ARRAY

a. The array `data` consists of four cells, each holding an integer value. The first cell, `data[0]`, is allocated but not used in this problem. After processing the entire file, the variable `data[n]` contains the number of votes for candidate n. We could have stored the information for candidate 1 in position 0, candidate 2 in position 1, and so forth, but the code is easier to follow if we can use a direct correspondence.

data

0	75	32	19
data[0]	data[1]	data[2]	data[3]

b. The value of `vote` is used to increment the appropriate cell of the array by +1.

2. A second example counts the occurrence of each alphabet letter in a text file.

Program 17-3

```
import chn.util.*;

public class CountLetters
{
  public static void main (String[] args)
  {
    FileInput inFile = new FileInput("sample.txt");

    int[] letters = new int[27]; // use positions 1..26
                                 //  to count letters
    int total = 0;
    char ch;

    while (inFile.hasMoreLines())
    {
      String line = inFile.readLine().toLowerCase();
      for(int index = 0; index < line.length(); index++)
      {
        ch = line.charAt(index);
        // line.charAt is from String clas. NOT in AP subset

        if ('a' <= ch && ch <= 'z')  // if we have a letter...
        {
          letters[ch - 96]++;  // if ch == 'a', 97-96 = 1, etc.
          total++;
        }
      }
    }
    System.out.println("Count letters");
    System.out.println();
    ch = 'a';
    for (int loop = 1; loop <= 26; loop++)
    {
      System.out.println(ch + " : " + letters[loop]);
      ch++;
    }
    System.out.println();
    System.out.println("Total letters = " + total);
  }
}
```

CHAINING (L → R)

YIELDS A String WHICH CAN CALL THIS

ANOTHER FREQUENCY ARRAY

a. Each line in the text file is read in and then each character in the line is copied into ch. If ch is an uppercase letter, it is converted to its lowercase counterpart.
b. If the character is a letter, the ASCII value of the letter is adjusted to fit the range from 1-26. For example, if ch == 'a', the program solves 97 - 96 = 1. Then the appropriate cell of the array is incremented by one.
c. Again, position 0 in the array is not used to make the data processing easier.

THUS FAR, WE'VE DONE CALL BY VALUE.

NOW, WITH ARRAYS ITS CALL BY REFERENCE

E. Arrays as Parameters

See ArrayOps.java, *Example Program - Arrays as Parameters.*

1. The program *ArrayOps.java*, provides examples of passing arrays as parameters. Notice that the **final** integer constant `MAX = 6` is used to size the array in this program.

2. The `main` method declares an array named `data`. The array is initialized with the values 0...5 inside the `main` method.

CALL BY REFERENCE

3. The parameters of the `squareList` and `printList` methods are references to an array object. Any local reference to array `list` inside the `squareList` or `printList` methods is an alias for the array `data` inside of the `main` method. Notice that after the call of `squareList`, the values stored in array `data` in the `main` method have been permanently changed.

FORCED SOLUTION FOR CALL BY VALUE

4. When the `rotateList` method is called, the copy method of the `ArrayOps` class is invoked and the local array `listCopy` is created as a copy of the array `data` in the `main` method.

5. The `rotateList` method rotates the values one cell to the right, with the last value moved to the front of the list. A call to `printList` is made inside the `rotateList` method just before leaving the method. After returning to the `main` method, notice that the array `data` is unchanged.

F. Arrays and Algorithms

In the following list we introduce five important algorithms that are quite common in programs that analyze data in arrays. You will meet these again in later lessons and labs.

1. Insertion is a standard problem that must be solved for all data structures. Suppose an array had 10 values and an 11th value was to be added. We are assuming the array can store at least 11 values.

   a. If we could place the new value at the end, there would be no problem.
   b. But if the new value must be inserted at the beginning of the list in position 0, the other 10 values must be moved one cell down the list.

2. Deletion of a value creates an empty cell that probably must be dealt with. The most likely solution after deleting a value, is to move all values that are to the right of the empty spot one cell to the left.

3. A traversal of an array consists of visiting every cell location, probably in order. The visit could involve printing out the array, initializing the array, finding the largest or smallest value in the array, etc.

4. Sorting an array means to organize values in either ascending or descending order. These algorithms will be covered in depth in future lessons.

5. Searching an array means to find a specific value in the array. There are several standard algorithms for searching an array. These will be covered in future lessons.

SUMMARY/ REVIEW: Arrays are extremely useful data structures and you will have many opportunities (lots of labs!) to program with them. After spending a few days working with single dimension arrays, we will move on to multi-dimensional arrays.

ASSIGNMENT: Lab Exercise L.A.17.1, *Statistics*
Lab Exercise L.A.17.2, *Compact*