

APS 105 – Computer Fundamentals

Lab #6: Arrays

Fall 1999

(To be completed *before* your lab period, week of October 18.)

Objective: To gain experience with the use and manipulation of arrays.

1 Unique Integers

Write a program that prompts the user for a list of ten unique integers, and then prints the list on a single line in the order they were entered. If the user enters an integer that was previously entered, the program should print "Duplicate value." and prompt for the integer again. An example of the use of the program is given below. (*Hint: Use an array to store the integers; check this array for duplicates as each new integer is entered.*)

```
Enter 10 unique integers.
Integer 1: 123
Integer 2: 456
Integer 3: 789
Integer 4: -1
Integer 5: -2
Integer 6: 456
Duplicate value.
Integer 6: -1
Duplicate value.
Integer 6: 2
Integer 7: 8
Integer 8: 456
Duplicate value.
Integer 8: 12
Integer 9: -1
Duplicate value.
Integer 9: 10
Integer 10: 9
123 456 789 -1 -2 2 8 12 10 9
```

1

2 The Game of Adjacency

For this part of the lab, you are to write a program which permits two players to play the game of *adjacency*. The game is played using X's and O's on an 8 by 8 square board, initially set up as in the following diagram.

```
____ OO 8
____ OO 7
____ 6
____ 5
____ 4
XX ____ 3
XX ____ 2
XX ____ 1
12345678
```

The squares on the board are identified using a standard Cartesian co-ordinate system, with the *x* co-ordinate identifying the column and the *y* co-ordinate the row. The bottom left corner would be at location 1,1 and the top right corner is at 8,8.

Player X begins the game by placing an X on one of the *empty* squares. All O's that are on squares adjacent to the selected square are considered *captured* and replaced with X's. For example, should player X put his (or her) first piece on co-ordinate 6,7 the resulting board configuration would be:

```
____ XO 8
____ XXO 7
____ 6
____ 5
____ 4
XX ____ 3
XX ____ 2
XX ____ 1
12345678
```

A square is considered adjacent if it is one square above, below, to the side or diagonal from the selected square.

Play continues with player O selecting a square, and play then alternating between the players until the board is filled. After every move, the new board configuration should be printed on the screen. It would be helpful if you also printed the coordinates and perhaps even the current score. You should also indicate whose turn it is.

Once no more empty squares remain, the program must count the number of X's and O's to find the player with the most squares, who is announced as the winner. Of course, since there are an even number of squares, a tie may occur.

Your program must prompt the users to input their moves until enough valid co-ordinate pairs have been entered to fill the board. Naturally, the program must be robust and identify invalid co-ordinates. You should precede each input with an appropriate prompt. You should catch an invalid coordinate value immediately and ask the user to reenter it right away.

In order to make it easier to mark your program, when the co-ordinate pair (0,0) is entered, you are to end the game early and announce the winner based on the current board condition.

Your game should be in a class named *Adjacency*. To assist you in structuring your program, you can use the file `/share/copy/aps105/Lab6.java` as a starting point. This file contains a set of *stub*

2

methods that you can complete to obtain a solution to the lab. If you desire, you may add additional methods to those provided.

When testing your program, you will find it useful to exploit the operating system's *pipe* facility. A pipeline connects the output of one program to become the input of another. By using the `cat` command, you can print the contents of a file to the screen. Using a pipe, represented with the symbol `|`, you can send the output of `cat` into your Java application instead. For example, suppose you create the file, `moves` (using your editor) with the following contents:

```
7
6
3
3
0
0
```

you can then use this file to play the game with the command:

```
cat moves | java Adjacency
```

Note that the methods of the `StdIn` class read a complete line every time they are called, therefore requiring each individual input to be on a separate line. The example file contains three co-ordinate pairs, 7,6, 3,3 and the game terminating pair 0,0.

To use pipelining, you must copy a new version of `/share/copy/aps1.05/StdIn.java` to your current working directory. The version that you used for labs 1 through 5 does not support pipelining. Also, Java does not display the input when it is received through a pipe, therefore you will need to add an extra `println` statement to obtain output that matches the previous example.

The final board configuration, and subsequent scoring, for playing the game with this input would be:

```
Turn #2 (Player O)
What is the column? 3
What is the row? 3
```

```
OO 8
XX 7
  X 6
   5
   4
  O 3
XO 2
XX 1
12345678
```

```
Turn #3 (Player X)
What is the column? 0
What is the row? 0
```

```
Final Score
Player X: 6
Player O: 4
Player X wins!
```

3 Optional Part

If you're ambitious you can *optionally* extend the program to follow either rule 1 or 2 below, and perhaps rule 3:

1. A player may only choose a square if it is adjacent to one of the opponents pieces. If there is no square on the board which satisfies the above criteria, the computer must identify this and announce that the player must skip the turn, with play proceeding to the other player. **OR**
2. A player may only choose a square if it is adjacent to one of his own pieces. If there is no square on the board which satisfies the above criteria, the computer must identify this and announce that the player must skip the turn, with play proceeding to the other player.
3. *Optional rule:* rather than choosing a blank square, the player may choose to jump one of his existing pieces from the current spot on the board over one square (which may or may not be occupied) into an empty square in the horizontal, vertical, or diagonal directions. The origin of the jump becomes blank, and the squares adjacent to the destination are all captured as before. This rule should be used in conjunction with rule 1 or 2.

4 Optional Optional Part

Very ambitious students can implement the program to use graphics as an applet. If you choose to implement the game as an applet, be warned that you will need to use programming techniques that have not been covered in class.