Homework 5 Part 1 Solution

CS 1323, Spring 2015

1. (10 points) Use memory diagrams to trace the code below:

a)

int[] data;

data = new int[4];

for (int i=0; i<data.length; ++i)

data[i] = i\*i; Heap

|  |  |  |
| --- | --- | --- |
|  | **Address** | **Contents** |
| 0 | 1000 | 0 |
| 1 | 1001 | 1 |
| 2 | 1002 | 4 |
| 3 | 1003 | 9 |
| length | 1004 | 4 |
|  | 1005 |  |
|  | 1006 |  |

Stack Frame

|  |  |  |
| --- | --- | --- |
|  | **Address** | **Contents** |
| data | 100 | 1000 |
|  | 101 |  |
|  | 102 |  |

b)

int[] data = {1, 3, 5, 7, 9};

int[] copy = data;

copy[3] = 2;

copy[4] = 6;

Heap

|  |  |  |
| --- | --- | --- |
|  | **Address** | **Contents** |
| 0 | 1000 | 1 |
| 1 | 1001 | 3 |
| 2 | 1002 | 5 |
| 3 | 1003 | ~~7~~ 2 |
| 4 | 1004 | ~~9~~ 6 |
| length | 1005 | 5 |
|  | 1006 |  |

Stack Frame

|  |  |  |
| --- | --- | --- |
|  | **Address** | **Contents** |
| data | 100 | 1000 |
| copy | 101 | 1000 |
|  | 102 |  |

1. (30 points; 5 points each part for the memory diagram, and 1 point for the values in the array) For each code fragment below, show a memory diagram that traces the program’s execution and give the value in the array data and the int variable size (part e) only) after the method has executed.

a)

//calling method

int[] data = {1, 3, 5, 7, 9};

method(data);

public static void method(int[] source)

{

source[0] = 2;

} Heap

|  |  |  |
| --- | --- | --- |
|  | **Address** | **Contents** |
| 0 | 1000 | ~~1~~ 2 |
| 1 | 1001 | 3 |
| 2 | 1002 | 5 |
| 3 | 1003 | 7 |
| 4 | 1004 | 9 |
| length | 1005 | 5 |
|  | 1006 |  |
|  | 1007 |  |
|  | 1008 |  |
|  | 1009 |  |
|  | 1010 |  |
|  | 1011 |  |
|  | 1012 |  |

Main Stack Frame

|  |  |  |
| --- | --- | --- |
|  | **Address** | **Contents** |
| data | 101 | 1000 |
|  | 102 |  |
|  | 103 |  |

Method Stack Frame

|  |  |  |
| --- | --- | --- |
|  | **Address** | **Contents** |
| source | 200 | 1000 |
|  | 201 |  |
|  | 202 |  |

b)

//calling method

int[] data = {1, 3, 5, 7, 9};

method(data);

public static void method(int[] source)

{

source = new int[2];

source[0] = 2;

source [1] = 4;

}

Heap

|  |  |  |
| --- | --- | --- |
|  | **Address** | **Contents** |
| 0 | 1000 | 1 |
| 1 | 1001 | 3 |
| 2 | 1002 | 5 |
| 3 | 1003 | 7 |
| 4 | 1004 | 9 |
| length | 1005 | 5 |
| 0 | 1006 | ~~0~~ 2 |
| 1 | 1007 | ~~0~~ 4 |
| length | 1008 | 2 |
|  | 1009 |  |
|  | 1010 |  |
|  | 1011 |  |
|  | 1012 |  |

Main Stack Frame

|  |  |  |
| --- | --- | --- |
|  | **Address** | **Contents** |
| data | 101 | 1000 |
|  | 102 |  |
|  | 103 |  |

Method Stack Frame

|  |  |  |
| --- | --- | --- |
|  | **Address** | **Contents** |
| source | 200 | ~~1000~~ 1006 |
|  | 201 |  |
|  | 202 |  |

c)

//calling method

int[] data = {1, 3, 5, 7, 9};

method(data);

public static void method(int[] source)

{

source[0] = 2;

source [1] = 4;

source = new int[2];

} Heap

|  |  |  |
| --- | --- | --- |
|  | **Address** | **Contents** |
| 0 | 1000 | ~~1~~ 2 |
| 1 | 1001 | ~~3~~ 4 |
| 2 | 1002 | 5 |
| 3 | 1003 | 7 |
| 4 | 1004 | 9 |
| length | 1005 | 5 |
| 0 | 1006 | 0 |
| 1 | 1007 | 0 |
| length | 1008 | 2 |
|  | 1009 |  |
|  | 1010 |  |
|  | 1011 |  |
|  | 1012 |  |

Main Stack Frame

|  |  |  |
| --- | --- | --- |
|  | **Address** | **Contents** |
| data | 101 | 1000 |
|  | 102 |  |
|  | 103 |  |

Method Stack Frame

|  |  |  |
| --- | --- | --- |
|  | **Address** | **Contents** |
| source | 200 | ~~1000~~ 1006 |
|  | 201 |  |
|  | 202 |  |

d)

//calling method

int[] data = {1, 3, 5, 7, 9};

data = method(data);

public static int[] method(int[] source)

{

source = new int[2];

source[0] = 2;

source [1] = 4;

return source;

} Heap

|  |  |  |
| --- | --- | --- |
|  | **Address** | **Contents** |
| 0 | 1000 | 1 |
| 1 | 1001 | 3 |
| 2 | 1002 | 5 |
| 3 | 1003 | 7 |
| 4 | 1004 | 9 |
| length | 1005 | 5 |
| 0 | 1006 | ~~0~~ 2 |
| 1 | 1007 | ~~0~~ 4 |
| length | 1008 | 2 |
|  | 1009 |  |
|  | 1010 |  |
|  | 1011 |  |
|  | 1012 |  |

Main Stack Frame

|  |  |  |
| --- | --- | --- |
|  | **Address** | **Contents** |
| data | 101 | ~~1000~~ 1006 |
|  | 102 |  |
|  | 103 |  |

Method Stack Frame

|  |  |  |
| --- | --- | --- |
|  | **Address** | **Contents** |
| source | 200 | ~~1000~~ 1006 |
|  | 201 |  |
|  | 202 |  |

e)

//calling method

int[] data =new int[10];

int size = 3;

for (int i=0; i<size; ++i)

data[i] = 2\*i + 1;

method(data, size, 9);

public static void method(int[] source, int size, int value)

{

source[size] = value;

size = size + 1;

}

Heap

|  |  |  |
| --- | --- | --- |
|  | **Address** | **Contents** |
| 0 | 1000 | ~~0~~ 1 |
| 1 | 1001 | ~~0~~ 3 |
| 2 | 1002 | ~~0~~ 5 |
| 3 | 1003 | 0 9 |
| 4 | 1004 | 0 |
| 5 | 1005 | 0 |
| 6 | 1006 | 0 |
| 7 | 1007 | 0 |
| 8 | 1008 | 0 |
| 9 | 1009 | 0 |
| length | 1010 | 10 |
|  | 1011 |  |
|  | 1012 |  |

Main Stack Frame

|  |  |  |
| --- | --- | --- |
|  | **Address** | **Contents** |
| data | 101 | 1000 |
| size | 102 | 3 |
|  | 103 |  |

Method Stack Frame

|  |  |  |
| --- | --- | --- |
|  | **Address** | **Contents** |
| source | 200 | 1000 |
| size | 201 | ~~3~~ 4 |
| value | 202 | 9 |

1. (10 points; 5 points each) Trace the following nested loops using the table on the right. Show every time a variable is changed—including the last change.

|  |  |  |
| --- | --- | --- |
| **sum** | **count** | **index** |
| 0 | 0 | 0 |
| 3 |  | 1 |
| 4 |  | 2 |
| 13 |  | 3 |
| 20 |  | 4 |
| 23 | 1 | 0 |
| 24 |  | 1 |
| 33 |  | 2 |
| 40 |  | 3 |
|  |  | 4 |
| 43 | 2 | 0 |
| 44 |  | 1 |
| 53 |  | 2 |
| 60 |  | 3 |
|  |  | 4 |
|  | 3 |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

int[] data = {3,1,9,7}; // constructs and initializes an array

int sum = 0;

for (int count = 0; count < 3; ++count)

{

for (int index = 0; index < data.length; ++index)

{

sum = sum + data[index];

}

}



|  |  |  |
| --- | --- | --- |
| **sum** | **index** | **count** |
| 0 | 0 |  |
| 3 |  | 1 |
| 6 |  | 2 |
| 9 |  | 3 |
|  |  | 4 |
|  | 1 |  |
| 10 |  | 2 |
| 11 |  | 3 |
|  |  | 4 |
|  | 2 |  |
| 20 |  | 3 |
|  |  | 4 |
|  | 3 |  |
|  |  | 4 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

int[] data = {3, 1, 9, 7};

int sum = 0;

for (int index = 0; index < data.length; ++index)

{

for (int count = index+1; count < data.length; ++count)

{

sum = sum + data[index];

}

}

1. (26 points; 9 points each for a) and b), 8 points for c)) This problem is a design problem. **Do not implement the program.**

Oklahoma has been having earthquakes lately. You’ve been asked to write a program that will read the data from a file of all earthquakes that is kept by the United States Geological Survey and calculate how many earthquakes of a given severity occurred in a given location this year. The file lists the location and severity of all earthquakes that were registered in the U.S. The getDataFromFilesFromLocation method should result in the severity array containing only data from a single given location.

a) If we wish to use perfect sized arrays (i.e. figure out how big the array should be first, then allocate it), what should the signatures of the methods be? A method signature includes the return type, the method name, and the parameters.

You will read data from the files into an array of doubles declared below. This array should contain only earthquakes from the chosen location.

double[] severity;

double[] getDataFromFilesFromLocation (String filename) // reads data from file into an array

double[] findEarthQuakesOfSeverityAtLocation(double[] severity, double threshold)

//uses all of the data stored in an array from a given location to create another array that contains only the earthquakes the exceed a certain severity

double findMeanSeverityAtLocation (double[] severity)

// uses the data stored in an array from a given location to find the mean (average) severity of the recorded earthquakes

b) If we wish to use super size arrays (i.e. assume that there aren’ t more than 10000 earthquakes in a year), what should the method signatures be?

You will read data from the files into an array of doubles declared below. This array should contain all earthquakes from the given location. The variable size will keep track of the active part of the super size array.

double[] severity;

int size;

int getDataFromFilesFromLocation(double[] severity, String filename)

// reads data from file into an array

Int findEarthQuakesOfSeverityAtLocation(double[] severity, int size, int[] biggestEarthquakes, double threshold)

//uses all of the data stored in an array from a given location to create another array that contains only the earthquakes the exceed a certain severity

double findMeanSeverityAtLocation(double[] severity, int size) // uses the data stored in an array from a given location to find the mean (average) severity of the recorded earthquakes

c) Where should the arrays in a) and b) be constructed? You may assume that the three methods above are called directly from the main program.

For part a): In the methods

For part b): Before the methods are called (usually in the main program, but it could be in another method too).

1. (10 points) Trace the execution of selection sort, using the algorithm presented in class (no other version will be accepted). Show each data movement on a separate line in the table. The tables are given a default size, which may be either too big or too small. If it’s too big, delete extra rows. If it is too small, add extra rows.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 4 | 2 | 1 | 3 | 5 | 9 | 7 |
| 1 |  | 4 |  |  |  |  |
|  |  | 3 | 4 |  |  |  |
|  |  |  |  |  | 7 | 9 |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |