Lab1

Gift Langa (1043882) *, Pierre Nayituriki (1045397) †,
Zanele Malinga (0706885N) ‡, Mokhulwane Nchabeleng (1117206) §
February 22, 2018
ELEN4020: Data Intensive Computing
School of Electrical and Information Engineering, University of the Witwatersrand, Johannesburg 2050, South Africa

I. Introduction

This report discusses the implementation of a program in C that takes in a K-dimensional array to perform three procedures on it. The first procedure is to initialize the K-dimensional array to zeros, the second procedure is to set ten percent of the array elements to ones and the third process selects random elements and makes five percent of the total array elements to print their coordinates and values.

II. First Procedure

The first procedure initializes all the elements in the array to zero. The function which performs this task is required to work for any given array regardless of its dimensions. To perform this, any array given to the function has to be treated as a one dimensional array by making use of a pointer which points to the first element of the array and then used to iterate to the last element. The $setKArrayToZeros$ performs this task by taking in two parameters - a pointer pointing to the first element of the given array which is ultimately used as an iterator and a second parameter containing the number of elements in the array to determine the index of the last element and to prevent any segmentation fault errors. These two parameters of the function are used together with a for loop to iterate from the first element to the last element whilst changing each element the pointer points to, to zero. The modified array is sent to the $printArray$ function to observe if the function behaves as intended. Figure 1 in the appendix depicts the first procedure.

III. Second Procedure

The primary objective of the second procedure is to uniformly transform ten percent of the array elements from zeros to ones. The procedure of this function is similar to the first procedure in code and also has the same parameters. The second procedure is achieved by also passing in a pointer to the first element of the array and the number of elements as parameters to the $setTenPercentToOnes$ function. There is a preprocessor definition of a global variable that is used in the $setTenPercentToOnes$ function to allow for the correct number of array elements to be transformed to ones. The variable is made global because its value is not required to change anywhere in the program. The $maxPoints$ variable is globally defined to account for ten percent of the total array elements to be changed to a value of one. The 'for' loop simply loops through the array by using the pointer parameter to do the zero to one transformation of array elements. There is an existing $printArray$ function that is used to output this procedure to verify its correctness. Figure 2 in the appendix presents a flow diagram of this procedure.

IV. Third Procedure

The third procedure of the lab was to randomly choose five percent of the elements of the array and print out the coordinates of each element and its value. Figure 3 shows the flow diagram of the code implementation of procedure three within the function $findFivePercent$. The function takes in four parameters which are; a pointer to the first element of the array 'A', the number of elements in the array 'A', a pointer which points to an array that contains the value of the bounds and the size of array 'A'. The function uses two pointers, one which points to the randomly selected element and the other is used to calculate the position of the selected element by iterating through the array in steps depending on the value of the bounds of the array. The steps are calculated using equation 1.

$$step = NumElements/(*(bounds + i)) \tag{1}$$

where:
$step$ is the step in which pointer 2 is traversing through the array 'A'. $NumElements$ is the number of elements within one of the dimensions of array 'A'. $*(bounds + i)$ is the value of one of the dimensions of array 'A'. i is changed by a 'for' loop and is incremented by one until all the elements in the bounds of the array are used up. A variable 'count' is used to track the coordinates of the element in array 'A' as pointer and traverses the array 'A'. 'count' is incremented when pointer 2 is

incremented by the $step$ and pointer 2's value is less than pointer 1. If the latter condition is not met, 'count' is reset to zero. $NumElements$ is set to the current value of $step$, $step$ is re-calculated using equation 1. This cycle is done until all the values in the bounds of the array are used up and count is printed at each cycle.

V. Conclusion

The objective of all three procedures has been successfully met with a print function that displays the result of the arrays regardless of there dimensions. Several dimensions of arrays which are passed into all the functions from the 'main' function are validated by passing in the arrays one at a time. This implies that each array that is called in main must be called by itself by commenting out all the other array calls.

APPENDIX A
FIGURES AND DIAGRAM

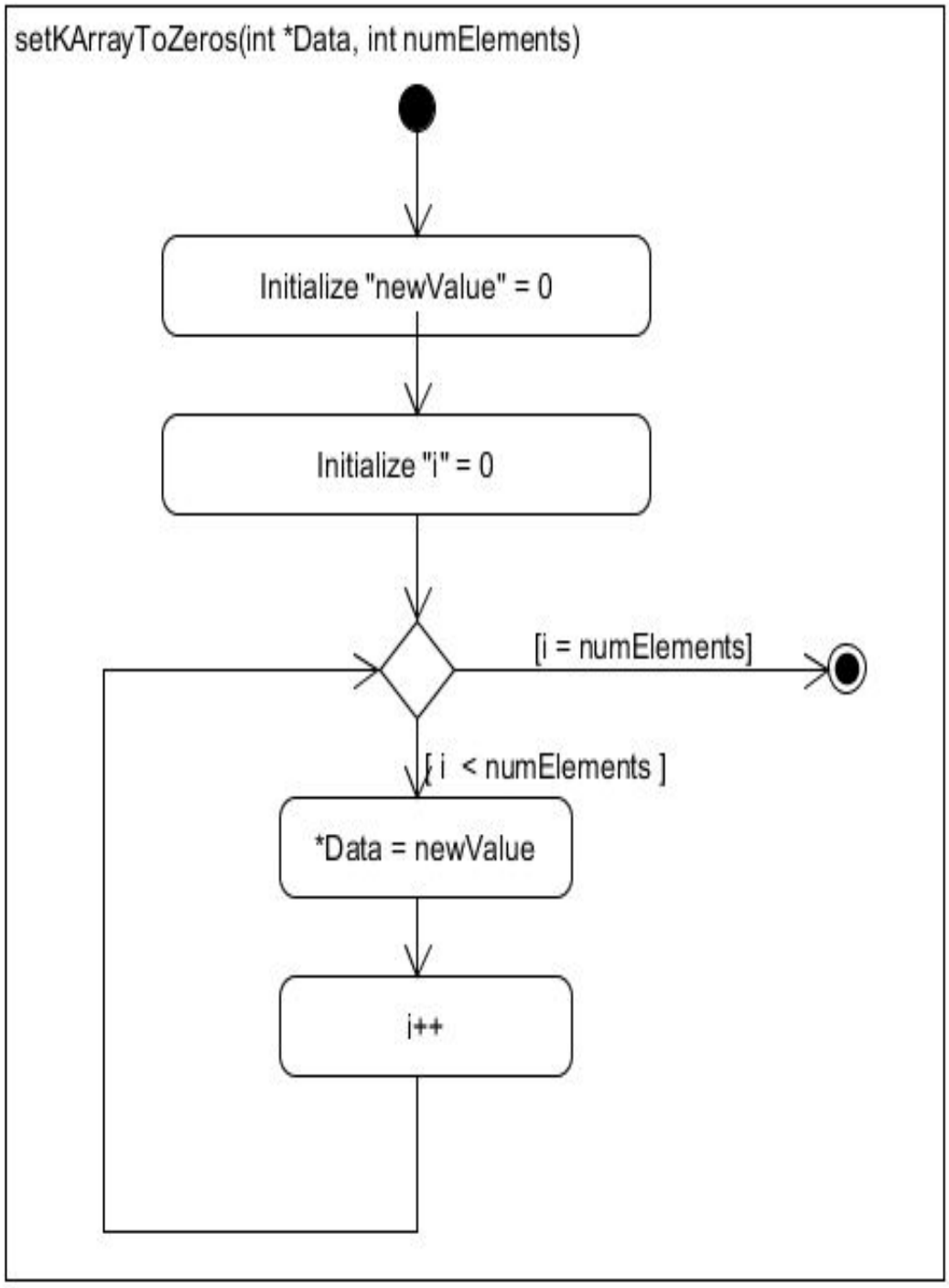

Fig. 1. The flow diagram of procedure 1

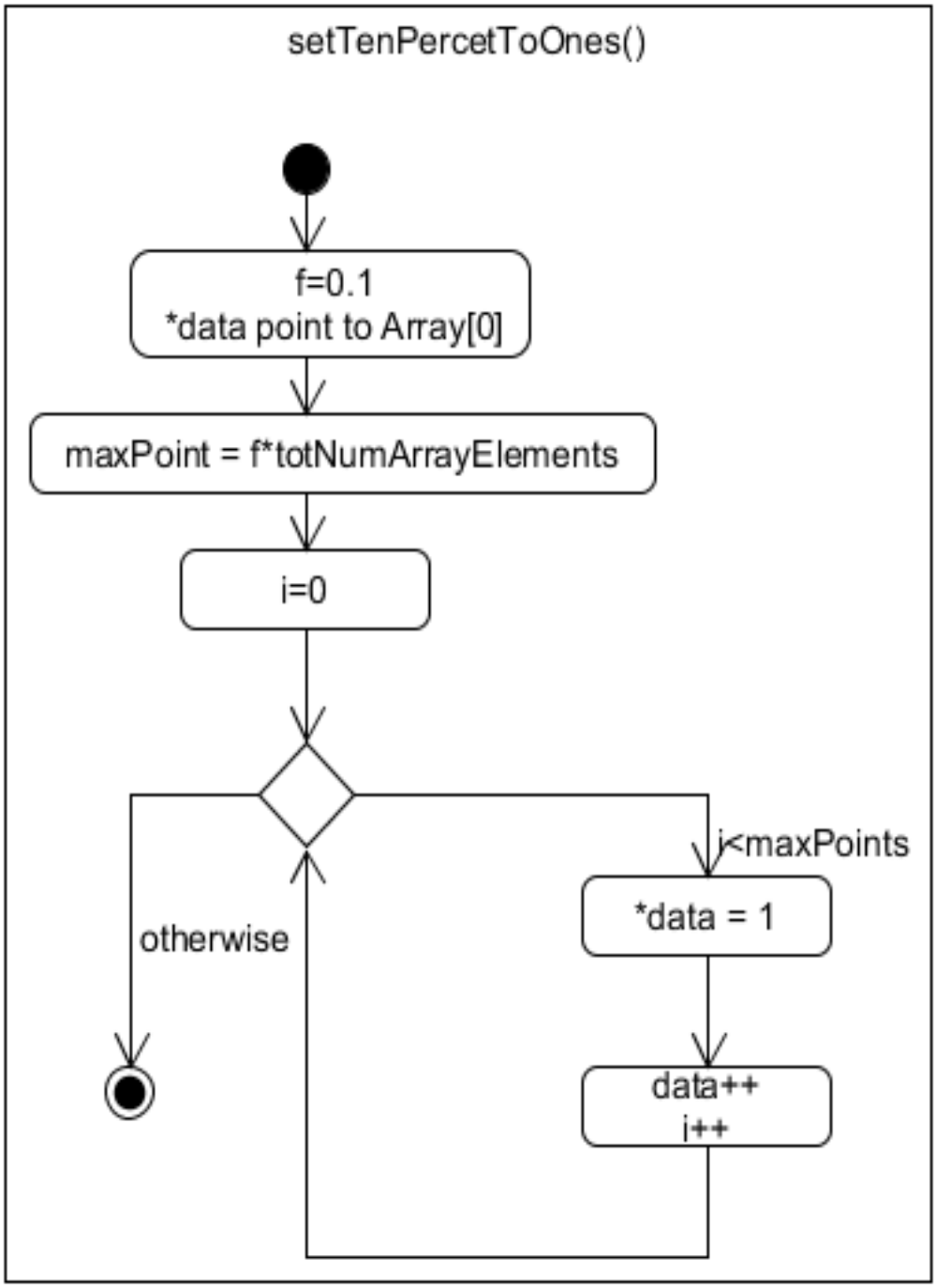

Fig. 2. The flow diagram of procedure 2

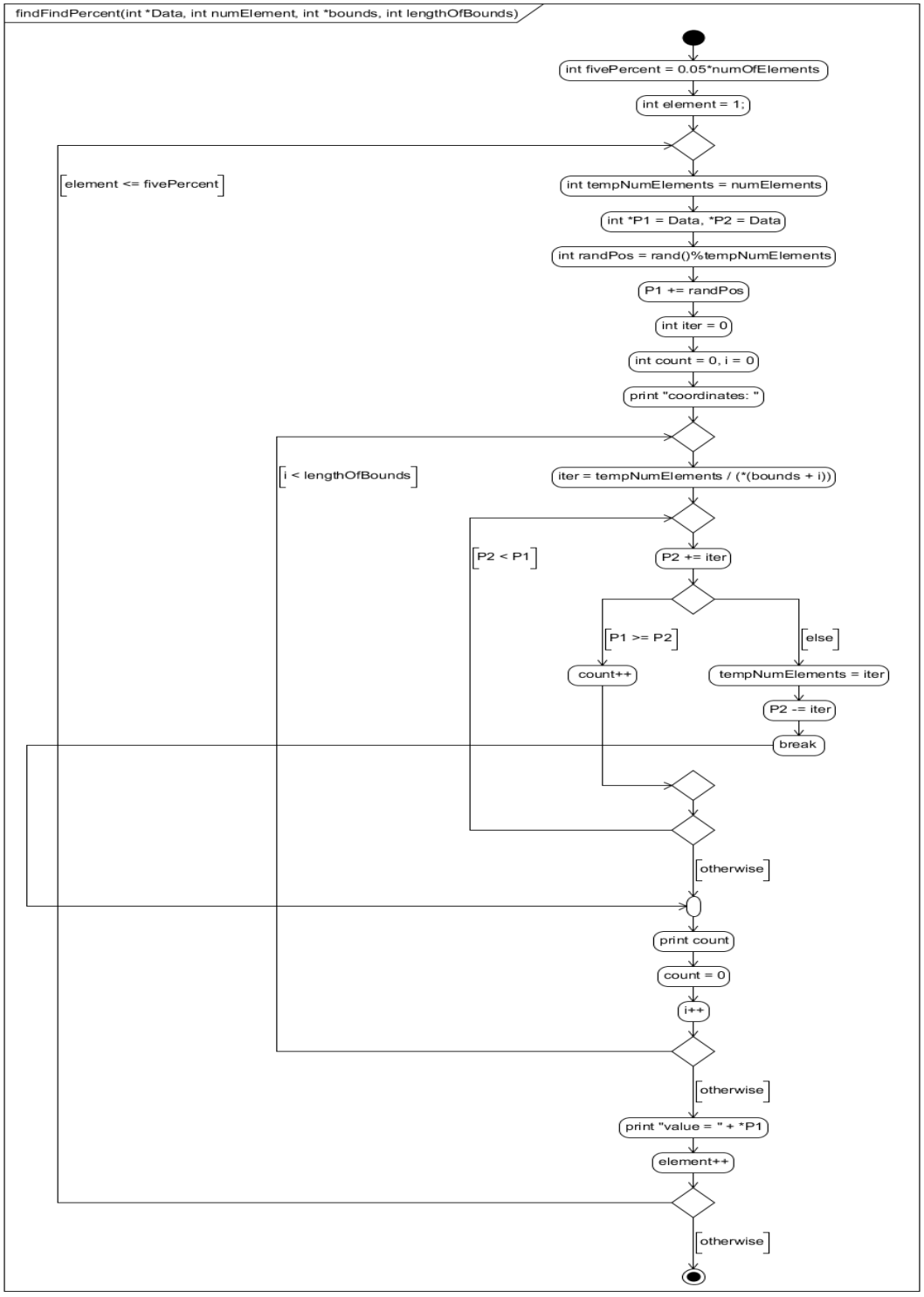

Fig. 3. the flow diagram of procedure 3