Labolatory Exercise no: 1

Sizwe Lekoba (793314)*, Phiwe Mbongwa (606352)†, Ndabezinhle Ndlovu (545276)‡, Tshembani Sibuyi (838214)
School of Electrical and Information Engineering, University of the Witwatersrand, Johannesburg 2050, South Africa
ELEN4020: Data Intensive Computing in Data Science

I. SOLUTION

A. First procedure: initializing elements to zero

This function takes two pointers as its arguments. The first argument points to a multi-dimensional array and the second argument point to an array that stores the bounds of the array. The total number of elements stored in the array is determined from the array with bounds. C compilers use row major ordering for multidimensional arrays. So once the size is acquired iteration is possible through the one dimensional array as seen in memory. A for loop is used to iterate through the array and initializes each element to 0.

Algorithm 1 Procedure 1

```
1:  procedure (ZerosArrayFunct(array, bounds))
2:      Array size ← set to 1
3:      temp1 ← bounds
4:      temp2 ← array
5:  while(bounds != 0)
6:      size*bounds
7:      bounds ← bounds + 1.
8:  end while
9:      bounds ← temp1
10: for(i = 0 to size-1)
11:     array[i] ← 0
12:     array ← array + 1
13: end for
14:     array ← temp2
15: end
```

B. second procedure: initializing 10% of the array to 1

This function declaration is identical to the first procedure, with the same arguments and follows the same path for calculating the size of the array. a for loop is implemented which iterates through the array and initializes only 10% of the array elements to 1.

C. third procedure: Randomly choosing 5% of the array to be printed

This function takes the same arguments as the first two procedures. The function begins by calculating the dimensions of the array and the total size of the array. An array that stores the element coordinates. A for loop is implemented to iterate through 5% of the array size and generates the coordinates of any random function. The randomly generated coordinates are used to determine the element from the 1 dimensional array passed into the function. Both the coordinates of the element and the contents in the element are printed.

Algorithm 2 Procedure 2

```
1:  procedure ( onesArrayFunct(array, bounds))
2:      Array size ← set to 1
3:      temp1 ← bounds
4:      temp2 ← array
5:  while(bounds != 0): size = size*bounds
6:      bounds ← bounds + 1.
7:      end while.
8:      bounds ← temp
9:  for(i =0 to size*0.1):
10:     array[i] ← 1
11:     array ← array + 1
12:     end for.
```

D. Main Program: Dynamically Generating the arrays and allocating memory

Dynamic allocation of memory for arrays is achieved by using the calloc() function which takes in the arguments of the size of the data type of the array and the number of elements in the array which is the product of the bounds of the array.

Algorithm 3 Procedure 3

1: **procedure** (ThirdFunct(array, bounds))
2: $n \leftarrow 0$
3: $mult \leftarrow 1$
4: $temp1 \leftarrow bounds$
5: $temp2 \leftarrow array$
6: *while(bounds != 0)*: *size = mult*bounds*
7: $n \leftarrow n + 1.$
8: $bounds \leftarrow bounds + 1.$
9: **end** *while*.
10: *bounds* $\leftarrow$ temp
11: *initialize array of coordinates*
12: $five \leftarrow 0.05 * mult.$
13: *for(i =0 to five)*:
14: *for(j = 0 to n)*:
15: *coordinates[j]* $\leftarrow$ random number of bounds
16: $bounds \leftarrow bounds + 1$
17: **end** *for*.
18: $bounds \leftarrow temp.$
19: $index \leftarrow numberfromarray.$
20: $c \leftarrow 0.$
21: *while(c is less than index)*:
22: $arrray \leftarrow array + 1.$
23: $c \leftarrow c + 1.$
24: **end** *while*.
25: *for(i = 0 to n)*:
26: *print(coordinates)*
27: **end** *for*.
28: *print(value)*
29: **end** *for*