Lab04 Report

uscheed

February 2019

1 Introduction

1. Use the `listings` package to include your output (`output_pt_a_vector.txt`) in your pdf. You will need to copy `output_pt_a_vector.txt` to the reports directory.

   ANSWER:

```
Test 0: Pass
Test 1: Pass
Test 2: Pass
Test 3: Pass
Test 4: Pass
Test 5: Pass
Test 6: Pass
Test 7: Pass
Test 8: Pass
Test 9: Pass
Test 10: Pass
Test 11: Pass
Test 12: Pass
Test 13: Pass
Test 14: Pass
Test 15: Fail
Test 16: Pass
Some Tests: Fail
```

2. For each of the following use-cases, indicate if the specified array should be allocated on the stack, the heap, or either. Explain your selection.

   (a) An array of integers length 10 in a function that is called a small number of times.

   - A stack should be used because it's used within one function and its length is very small.

   (b) An array of doubles of length 3, where $\sim 10^3$ instances exist and frequently used in the program.

   - A heap should be used because it's used throughout the whole program and the number of instances is fairly large.

   (c) An array of doubles of length 3, where $\sim 10^4$ instances exist and frequently used in the program.

   - A heap should be used because it's used throughout the whole program and the number of instances is fairly large.

   (d) An array of doubles of length 3, where $\sim 10^5$ instances exist and frequently used in the program.

   - A heap should be used because it's used throughout the whole program and the number of instances is very large.

   (e) An array of doubles of length 3, where $\sim 10^6$ instances exist and frequently used in the program.

- A heap should be used because it's used throughout the whole program and the number of instances is very large.

(f) An array of doubles of length 3, where $\sim 10^8$ instances exist and frequently used in the program.

- A heap should be used because it's used throughout the whole program and the number of instances is very large.

(g) An array of floats of length 10,000, to be used throughout the whole program.

- A heap should be used because it's used throughout the whole program its length is very large.

(h) An array of floats of length 10,000, to be used in a single function.

- A heap or stack can be used, but a heap is preferred because even though the scope of it is small, its length is fairly large.

ANSWER: Answer is bulleted.

3. In C, there is no mechanism to see if a pointer points to heap memory that has already been allocated, so we cannot be sure that we do not re-allocate an array. How can we code defensively to ensure that this does not happen?

   ANSWER: We save the pointer as a variable and free the memory after we're done using it.

4. Other than the instructor or TAs, who did you receive assistance from on this assignment?

   ANSWER: Lilian

1. `matrixTest`

   - Use the `listings` package to include your `matrixTest` output in the pdf.
   - For each bug, use the `listings` package to display the original line of code with the error, as well as the fix. Describe the error.

   ANSWER:

```
Testing dotProd
vecA = {1.000000, 2.000000, 3.000000}
vecB = {4.000000, 5.000000, 6.000000}
vecA*vecB = 32.000000
Test Passed!


Testing matrixVecProd with identity matrix
A =
1.000000 0.000000 0.000000
0.000000 1.000000 0.000000
0.000000 0.000000 1.000000
x = {1.000000, 2.000000, 3.000000}
b = {1.000000, 2.000000, 3.000000}
Test passed!


Testing matrixVecProd a short and fat matrix
A =
1.000000 2.000000 3.000000
4.000000 5.000000 6.000000
x = {1.000000, 2.000000, 3.000000}
b = {14.000000, 32.000000}
Test passed!
```

```
Testing matrixVecProd a short and fat matrix
A =
1.000000  2.000000
3.000000  4.000000
5.000000  6.000000
x = {1.000000, 2.000000}
b = {5.000000, 11.000000, 17.000000}
Test passed!
```

```
   double dotProd = 0;

-  for(int i=0; i<=n; i++){
+  for(int i=0; i<n; i++){
     dotProd += x[i]*y[i];
   }
```

```
 void matrixVecProd(int m, int n, double* A, double* x, double* b){

   for(int i=0; i<m; i++){
-    b[i] = dotProd(n, A+i*m, x);
+    b[i] = dotProd(n, A+i*n, x);
   }
 }
```

```
   double* c = vecDiff(m, b, ans);

-  if(norm(m, b)<tol){
+  if(norm(m, c)<tol){
     printf("Test passed!\n");
   }
   else{
```

The for loop has an equals sign as a parameter but there shouldn't be one because it's accessing unallocated areas of an array.
The formula to calculate the dot product was incorrect; C is a row ordered language so n should be used not m.
When testing the norm, b was used instead of c. I knew c should have been used because c was used for the same test statement further up in the file. Plus, c was defined but not used for anything within its scope.

2. `fibonacci`

   - Use the `listings` package to include your `fibonacci` output in the pdf.
   - For each bug, use the `listings` package to display the original line of code with the error, as well as the fix. Describe the error.

   ANSWER:

```
the first 32 fibonacci numbers are:
0: 1
1: 1
2: 2
3: 3
```

4: 5
5: 8
6: 13
7: 21
8: 34
9: 55
10: 89
11: 144
12: 233
13: 377
14: 610
15: 987
16: 1597
17: 2584
18: 4181
19: 6765
20: 10946
21: 17711
22: 28657
23: 46368
24: 75025
25: 121393
26: 196418
27: 317811
28: 514229
29: 832040
30: 1346269
31: 2178309

```
int main (int argc, char **argv) {
    //initialize variables
    int array_size = 32;
-   int *nums = (int *) malloc(array_size);
+   int *nums = (int *) malloc(array_size*sizeof(int));
```

The memory wasn't allocated right, we forgot to multiply by the size of an int when instantiating the array.

3. `pascal`

   - Use the `listings` package to include your `pascal` output in the pdf.
   - For each bug, use the `listings` package to display the original line of code with the error, as well as the fix. Describe the error.

   ANSWER:

```
                                                  1
                                              1       1
                                          1       2       1
                                      1       3       3       1
                                  1       4       6       4       1
                              1       5      10      10       5       1
                          1       6      15      20      15       6       1
                      1       7      21      35      35      21       7       1
                  1       8      28      56      70      56      28       8       1
              1       9      36      84     126     126      84      36       9       1
          1      10      45     120     210     252     210     120      45      10       1
      1      11      55     165     330     462     462     330     165      55      11       1
  1      12      66     220     495     792     924     792     495     220      66      12       1
      1      13      78     286     715    1287    1716    1716    1287     715     286      78      13       1
  1      14      91     364    1001    2002    3003    3432    3003    2002    1001     364      91      14       1
      1      15     105     455    1365    3003    5005    6435    6435    5005    3003    1365     455     105      15       1
  1      16     120     560    1820    4368    8008   11440   12870   11440    8008    4368    1820     560     120      16       1
```

```
int main(int argc,char **argv) {
    //depth of triangle
-   char depth = 17;
-   char spacing = 5;
-   char spacing_start = 3;
-   char length = depth*depth;
+   int depth = 17;
+   int spacing = 5;
+   int spacing_start = 3;
+   int length = depth*depth;
```

The values were created as a char, not an int.

4. `array_sum`

   - Use the `listings` package to include your `array_sum` output in the pdf.
   - For each bug, use the `listings` package to display the original line of code with the error, as well as the fix. Describe the error.

   ANSWER:

   ```
   2.635288
   2.058612
   1.948012
   1.976691
   1.228937
   ```

   No bugs found.

5. `rotate_vector`

   - Use the `listings` package to include your `rotate_vector` output in the pdf.
   - For each bug, use the `listings` package to display the original line of code with the error, as well as the fix. Describe the error.

   ANSWER:

   ```
   0.170828 0.749902 0.096372 0.870465 0.577304 0.785799 0.692194
   0.692194 0.170828 0.749902 0.096372 0.870465 0.577304 0.785799
   0.785799 0.692194 0.170828 0.749902 0.096372 0.870465 0.577304
   0.577304 0.785799 0.692194 0.170828 0.749902 0.096372 0.870465
   0.870465 0.577304 0.785799 0.692194 0.170828 0.749902 0.096372
   0.096372 0.870465 0.577304 0.785799 0.692194 0.170828 0.749902
   0.749902 0.096372 0.870465 0.577304 0.785799 0.692194 0.170828
   ```

```
    //fill rows of array with vector rotations
    for (int i = 0; i < vector_size; i++) {
-       for (int j = 0; j < vector_size; i++) {
+       for (int j = 0; j < vector_size; j++) {
            int index = (j+i)%vector_size;
            rotations[i*vector_size + index] = vector[j];
        }
    }
```

When filling the rows of array with vector rotations, the nested for loop incremented i instead of j, so I changed it from i++ to j++.