CMDA 3634
Lab 04 Report

Russell J. Hewett

February 18, 2019

Part A

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: Pass
Test 16: Pass
All Tests: Pass
```

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.

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

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

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

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

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

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

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

   ANSWER:

   (a) You should use the stack because the arrays are small and short lived.

(b) Either are acceptable, but the stack is probably a better choice if there is risk that the heap will become fragmented.

(c) Either are acceptable, but the stack is probably a better choice if there is risk that the heap will become fragmented, as long as the rest of the data used in the program is small.

(d) Either are acceptable, but the heap is getting more attractive, as the volume of data is starting to get large.

(e) The heap is required, as the volume of data will exceed the available stack memory. Care will need to be taken to avoid fragmentation. This pattern is probably a bad design.

(f) The heap is required, as the volume of data will exceed the available stack memory. Care will need to be taken to avoid fragmentation. This pattern is probably a bad design.

(g) The heap is preferred, as a single allocation will not fragment, but the stack will be sufficient.

(h) The stack is preferred here, as long as the function is not called frequently.

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 have to be vigilent that we initialize new pointers to `NULL`. Then, we can use custom allocators and deallocators that can check the status of the pointer before acting. As the C language has no support for this sort of protection, which is found in other languages like C++ and Fortran, we are required to code with discipline. Always match allocation to deallocation and always initialize pointers.

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

   ANSWER: No one.

Part B

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:

gdbOutput.txt

```
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 tall and skinny 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!
```

```
--- a/labs/lab04_sol/code/gdb/matrixProd.c
+++ b/labs/lab04_sol/code/gdb/matrixProd.c
@@ -7,7 +7,7 @@ double dotProd(int n, double* x, double* y){

    double dotProd = 0;

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

The loop counts 1 extra time, from 0 to N, when it should count from 0 to $N-1$.

```
--- a/labs/lab04_sol/code/gdb/matrixProd.c
+++ b/labs/lab04_sol/code/gdb/matrixProd.c
@@ -22,7 +22,7 @@ double dotProd(int n, double* x, double* y){
 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);
    }
 }
```

The calculation of the start of the row in the mat-vec is not correct. It assumes rows are length M and not N.

```
--- a/labs/lab04_sol/code/gdb/matrixTest.c
+++ b/labs/lab04_sol/code/gdb/matrixTest.c
@@ -118,7 +118,7 @@ void testShortFat(){
    double ans[2] = {14, 32};


-   double* c = vecDiff(n, b, ans);
+   double* c = vecDiff(m, b, ans);

    if(norm(m, c)<tol){
      printf("Test passed!\n");
```

For this test case, the vector b has length M, not N.

```
--- a/labs/lab04_sol/code/gdb/matrixTest.c
+++ b/labs/lab04_sol/code/gdb/matrixTest.c
@@ -155,7 +155,7 @@ void testTallSkinny(){
      }
    }

-   printf("Testing matrixVecProd a short and fat matrix\n");
+   printf("Testing matrixVecProd a tall and skinny matrix\n");

    printA(m, n, A);
```

The print text refers to the wrong test.

```
--- a/labs/lab04_sol/code/gdb/matrixTest.c
+++ b/labs/lab04_sol/code/gdb/matrixTest.c
@@ -170,7 +170,7 @@ void testTallSkinny(){

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

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

The test should check the norm of the error vector, c, not the right-hand side vector b.

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:

fibOutput.txt

```
the first 30 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
```

```
--- a/labs/lab04_sol/code/valgrind/fibonacci.c
+++ b/labs/lab04_sol/code/valgrind/fibonacci.c
@@ -4,7 +4,7 @@
 int main (int argc, char **argv) {
     //initialize variables
     int array_size = 30;
-    int *nums = (int *) malloc(array_size);
+    int *nums = (int *) malloc(array_size*sizeof(int));

     //seed with first two values
     nums[0] = 1;
```

The allocation if the `nums` array is missing the `sizeof`, so an incorrect number of bytes is allocated.

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:

pasOutput.txt

```
                                                  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
```

```
--- a/labs/lab04_sol/code/valgrind/pascal.c
+++ b/labs/lab04_sol/code/valgrind/pascal.c
@@ -7,7 +7,7 @@ int main(int argc,char **argv) {
     char depth = 17;
     char spacing = 5;
     char spacing_start = 3;
-    char length = depth*depth;
+    int length = depth*depth;

     //initial setup
     int *nums = (int *) malloc(length*sizeof(int));
```

The maximum number that a char can store is 255, but `depth*depth` is 289, so there is an overflow. We fix by using a number that can store more values.

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:

sumOutput.txt

```
2.635288
2.750806
2.318674
3.110816
2.368282
```

```
--- a/labs/lab04_sol/code/valgrind/array_sum.c
+++ b/labs/lab04_sol/code/valgrind/array_sum.c
@@ -6,7 +6,7 @@ int main(int argc, char **argv) {
     int arr_length = 5*5;

     float *sum = (float *) malloc(sum_length*sizeof(float));
-    float *arr = (float *) malloc(sum_length*sizeof(float));
+    float *arr = (float *) malloc(arr_length*sizeof(float));

     //seed array with random numbers
     srand48(0);
```

The `arr` array was not allocated with the correct length so the reads in constructing the sum access out of bounds.

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:

rotOutput.txt

```
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
```

```
--- a/labs/lab04_sol/code/valgrind/rotate_vector.c
+++ b/labs/lab04_sol/code/valgrind/rotate_vector.c
@@ -14,7 +14,7 @@ int main(int argc, char **argv) {

     //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];
         }
```

The loop increment uses the wrong variable, so `j` does not change and `i` increments faster than it should.