C-mini-challenges

Open in Visual Studio Code

For the following Mini-Challenges, you may use the Ubuntu VM set up for you. You will need to install gcc (and maybe valgrind), e.g., "sudo apt install gcc." The Cyber VM Infrastructure Notes should guide you on how to get started with the VMs.

I recommend creating a SSH key for the VM and associating it with your GitHub account. It takes a little bit of work but should be worth it in the long run. Guidance on how SSH key generation and GitHub association - generated by Claude and vetted by me - is available in the course contents section for this class.

And now, on to the Mini-Challenges:

1. Print “Hello, NAME” where NAME is input from the keyboard.
2. Implement Archimedes algorithm to estimate pi for inscribed/circumscribed polygons with n sides, up to 100, doubling n at each step, and time it.
3. Implement matrix – vector multiplication. Read in the following text file (mv.txt) which contains the matrix and vector to be multiplied. Print your answer to the screen and time the computation. The format of mv.txt is: line 1 contains numrows, numcols. The next numrows contains the rows of the matrix, each with numcols integer values. The next line contains the length of the vector. The next line contains the vector of that length.
4. Compare the speed of *,/,sqrt, sin operations/functions.
5. Use the attached code snippets as a basis for comparing the performance of row-major vs. column major computations. One snippet uses a static allocation for the array, the other allocates the array dynamically. Do a little experimentation with each approach. Vary the size of the square array from 128 X 128 on up, doubling it in size each time. Chart your results. Is there a difference in performance or behavior between static and dynamic? Between row-major and column-major? In terms of the latter, valgrind has been installed on your Linux VM, and its cachegrind tool facility may help provide some insights. Do some OSINT research to learn about valgrind....
6. Write a program that accepts a string input from stdio and sends it to a function that transforms it according a transposition function passed in to it as an argument. The function will print out the string, transform it, and then print out the result. The transposition function, you can assume, simply shuffles the existing characters in the string. Build a transposition function that reverses the string and apply it. Where appropriate and possible, use dynamic allocation and pointer arithmetic to get the job done.

C-Mini-Challenges Responses

1. For the Hello Name challenge

a. Did you try passing your name as an argument from the command line or did you use scanf? Why? I used `scanf` because it allows for an interactive prompt, which makes the program slightly easier to use for a casual user compared to passing command-line arguments via `argv`. However, in a strict High-Performance Computing environment, command-line arguments are generally preferred for automation.

b. How did you manage or allocate the strings? (Static or dynamic) I allocated the string statically by declaring a character array of a fixed size (`char name[100];`) on the stack.

2. For Archimedes algorithm

a. How did you time your program? I timed the program using the `clock_t` data type and the `clock()` function from the `<time.h>` library to record the start and end cycles. I then divided the difference by the `CLOCKS_PER_SEC` constant to calculate the elapsed time in seconds.

b. Were there any issues with precision and/or convergence that you noticed? Within the bounds of the assignment (stopping at n=128), there were no noticeable convergence issues; the approximation of pi became steadily more accurate. However, if the iterations were pushed significantly higher, floating-point precision limits and catastrophic cancellation would eventually cause the approximation to degrade.

3. For Matrix-vector multiplication

a. How did you allocate and access your matrix? I dynamically allocated a flattened 1D array on the heap using `malloc`. To access the equivalent of a 2D `matrix[i][j]` position, I calculated the offset using `i * cols + j`.

b. Were there any challenges in reading in the file? Once the correct standard library calls (`fopen`, `fscanf`) were implemented, reading the file was straightforward. The main challenge was ensuring proper validation, such as checking that the read vector length strictly matched the matrix column count before proceeding.

c. Was there anything special about the actual computation? To optimize the computation, I processed the dot products sequentially but used a local accumulator variable (`sum`) for the inner loop.

d. What was your strategy for timing? I used `clock_t` and `clock()` again, ensuring that the timer only wrapped the mathematical computation loops, entirely excluding the slow file I/O operations from the measurement.

4. For measuring the speed of arithmetic computations

a. What was your timing strategy? Because modern CPUs execute single arithmetic instructions too quickly to measure accurately, I placed the operation inside a loop running 10,000,000 times and timed the entire loop. I assigned the result to a `volatile double` variable which was necessary to prevent the compiler from optimizing and skipping the loop entirely, since the result is never printed or used.

b. Are all arithmetic operations created equal? No. Based on the timing results, the operations from fastest to slowest were: multiplication, division, square root, and sine.

5. For the row-major/column-major exercise

a. What did you observe about differences in program behavior in static vs dynamic allocation of arrays, and how do you explain it? Static allocation on the stack was slightly faster for large enough arrays because stack pointer arithmetic is highly optimized and localized. However, static Variable-Length Arrays (VLAs) are inherently limited by the small size of the stack and risk segmentation faults at larger sizes. Dynamic allocation on the heap required OS overhead to map the virtual memory to physical RAM (page faulting). Dynamic memory safely supported much larger array dimensions without crashing.

b. What did you observe about differences in program behavior in row-major vs. column major computations and how do you explain it? Row-major computations were significantly faster than column-major computations across all larger array sizes. This is fundamentally due to how C handles memory and how modern CPUs utilize caching. C stores 2D arrays in row-major order, meaning consecutive elements in a row are contiguous in physical RAM. When iterating in row-major order, pulling one value loads a full "cache line" of adjacent values into the CPU's ultra-fast L1 cache (spatial locality), resulting in very few cache misses due to `arr[i][j]` being next to `arr[i][j+1]` in memory. Conversely, column-major traversal jumps across memory addresses by the size of the row on every single iteration, constantly missing the L1 cache and forcing the CPU to repeatedly fetch data from much slower main memory.

6. For the string transform problem

a. What were some alternative implementation strategies could you entertain here? An alternative implementation could involve utilizing custom data structures, such as implementing a stack and queue from scratch. The characters could be pushed onto the stack sequentially and then popped off to reverse the order, whereas the character pointers could be pushed onto the queue and dequeued to preserve their order.

b. What programming hazards/pitfalls should be considered in your general approach? When manipulating strings via pointer arithmetic, hazards include failing to account for the null terminator (`\0`), accidentally swapping the newline character (`\n`) to the front of the string, or decrementing a pointer past the starting address of the array if the user inputs an empty string.

Note to self on ssh into VM

```
ssh student@10.30.248.205
```

C-Mini-Challenges Ouptut

```
student@CS-4373-SP26-C-DEV-05:~/c-mini-challenges-Pirate-Hunter-Zoro$ gcc
-g exercises.c -lm -o bin/main
student@CS-4373-SP26-C-DEV-05:~/c-mini-challenges-Pirate-Hunter-Zoro$
valgrind --tool=cachegrind ./bin/main 2>cache_report.txt
Enter a name:
Mikey
Hello, Mikey!


=============================================================
n: 128
C: 3.142224
I: 3.141277
```

```
Elapsed 0.000894 seconds...


============================================================
Resulting Product:
18817.000000 18431.000000 5967.000000 16686.000000 25429.000000
18817.000000 16835.000000 17175.000000 25000.000000 14060.000000
Time Elapsed: 0.000653 seconds


============================================================
Division Time for 10000000 Operations: 0.281879 seconds
Multiplication Time for 10000000 Operations: 0.239244 seconds
Square Root Time for 10000000 Operations: 0.590296 seconds
Sine Time for 10000000 Operations: 4.198955 seconds


============================================================
Static Array Size 128 by 128 Row Major Time: 0.001092 Seconds
Static Array Size 128 by 128 Column Major Time: 0.001145 Seconds
Dynamic Array Size 128 by 128 Row Major Time: 0.001055 Seconds
Dynamic Array Size 128 by 128 Column Major Time: 0.001006 Seconds

Static Array Size 256 by 256 Row Major Time: 0.002236 Seconds
Static Array Size 256 by 256 Column Major Time: 0.003275 Seconds
Dynamic Array Size 256 by 256 Row Major Time: 0.002388 Seconds
Dynamic Array Size 256 by 256 Column Major Time: 0.003404 Seconds

Static Array Size 512 by 512 Row Major Time: 0.009265 Seconds
Static Array Size 512 by 512 Column Major Time: 0.015685 Seconds
Dynamic Array Size 512 by 512 Row Major Time: 0.010129 Seconds
Dynamic Array Size 512 by 512 Column Major Time: 0.015079 Seconds


============================================================
Enter a string to be reversed:
terrestrial
Original String: terrestrial

Reverse String: lairtserret


student@CS-4373-SP26-C-DEV-05:~/c-mini-challenges-Pirate-Hunter-Zoro$ cat
cache_report.txt
==26767== Cachegrind, a high-precision tracing profiler
==26767== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote
et al.
==26767== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright
info
==26767== Command: ./bin/main
==26767==
==26767==
==26767== I refs:        1,760,351,868
```

```
student@CS-4373-SP26-C-DEV-05:~/c-mini-challenges-Pirate-Hunter-Zoro$ ls
 bin                       cachegrind.out.26659   cache_report.txt
loopsstaticshell.c   README.pdf
 cachegrind.out.26649   cachegrind.out.26759   exercises.c
mv.txt              'SoCs VM Notes.pdf'
 cachegrind.out.26658   cachegrind.out.26767   loopsdynamicshell.c
README.md
student@CS-4373-SP26-C-DEV-05:~/c-mini-challenges-Pirate-Hunter-Zoro$
cg_annotate cachegrind.out.26767
---------------------------------------------------------------------------
------
-- Metadata
---------------------------------------------------------------------------
------
Invocation:       /usr/bin/cg_annotate cachegrind.out.26767
Command:          ./bin/main
Events recorded:  Ir
Events shown:     Ir
Event sort order: Ir
Threshold:        0.1%
Annotation:       on

---------------------------------------------------------------------------
------
-- Summary
---------------------------------------------------------------------------
------
Ir____________________

1,760,351,868 (100.0%)  PROGRAM TOTALS

---------------------------------------------------------------------------
------
-- File:function summary
---------------------------------------------------------------------------
------
  Ir___________________________  file:function

< 1,108,930,523 (63.0%,  63.0%)  ./math/../sysdeps/ieee754/dbl-
64/s_sin.c:__sin_sse2

<   431,023,854 (24.5%,  87.5%)  /home/student/c-mini-challenges-Pirate-
Hunter-Zoro/exercises.c:
    400,000,104 (22.7%)            exercise_4
     31,018,107  (1.8%)            exercise_5

<    99,999,997  (5.7%,  93.2%)
./math/../sysdeps/x86/fpu/fenv_private.h:__sin_sse2

<    50,000,025  (2.8%,  96.0%)  ./math/./math/w_sqrt_compat.c:sqrt

<    40,001,572  (2.3%,  98.3%)  ???:
     40,001,560  (2.3%)            ???
```

```
<    30,000,015  (1.7%, 100.0%)  ./math/../sysdeps/ieee754/dbl-
64/e_sqrt.c:__sqrt_finite@GLIBC_2.15

--------------------------------------------------------------------------
-------
-- Function:file summary
--------------------------------------------------------------------------
-------
  Ir______________________________  function:file

> 1,208,930,520 (68.7%,  68.7%)  __sin_sse2:
  1,108,930,523 (63.0%)            ./math/../sysdeps/ieee754/dbl-
64/s_sin.c
     99,999,997  (5.7%)
./math/../sysdeps/x86/fpu/fenv_private.h

>   400,000,104 (22.7%,  91.4%)  exercise_4:/home/student/c-mini-
challenges-Pirate-Hunter-Zoro/exercises.c

>    50,000,025  (2.8%,  94.2%)  sqrt:./math/./math/w_sqrt_compat.c

>    40,001,560  (2.3%,  96.5%)  ???:???

>    31,018,107  (1.8%,  98.3%)  exercise_5:/home/student/c-mini-
challenges-Pirate-Hunter-Zoro/exercises.c

>    30,000,015  (1.7%, 100.0%)
__sqrt_finite@GLIBC_2.15:./math/../sysdeps/ieee754/dbl-64/e_sqrt.c

--------------------------------------------------------------------------
-------
-- Annotated source file: ./math/../sysdeps/ieee754/dbl-64/e_sqrt.c
--------------------------------------------------------------------------
-------
Unannotated because one or more of these original files are unreadable:
- ./math/../sysdeps/ieee754/dbl-64/e_sqrt.c

--------------------------------------------------------------------------
-------
-- Annotated source file: ./math/../sysdeps/ieee754/dbl-64/s_sin.c
--------------------------------------------------------------------------
-------
Unannotated because one or more of these original files are unreadable:
- ./math/../sysdeps/ieee754/dbl-64/s_sin.c

--------------------------------------------------------------------------
-------
-- Annotated source file: ./math/../sysdeps/x86/fpu/fenv_private.h
--------------------------------------------------------------------------
-------
Unannotated because one or more of these original files are unreadable:
- ./math/../sysdeps/x86/fpu/fenv_private.h

--------------------------------------------------------------------------
```

```
-------
-- Annotated source file: ./math/./math/w_sqrt_compat.c
--------------------------------------------------------------------------
-------
Unannotated because one or more of these original files are unreadable:
- ./math/./math/w_sqrt_compat.c

--------------------------------------------------------------------------
-------
-- Annotated source file: /home/student/c-mini-challenges-Pirate-Hunter-
Zoro/exercises.c
--------------------------------------------------------------------------
-------
Ir_______________

         .         #include <stdio.h>
         .         #include <math.h>
         .         #include <time.h>
         .         #include <stdlib.h>
         .
         .
//////////////////////////////////////////////////////////////////////////
///////////////////////////////
         .
         7 (0.0%)  void exercise_1() {
         .             // Read and print name
         .             char name[100];
         3 (0.0%)      printf("Enter a name:\n");
         .             // NOTE - do not enter in the address &name; name
is already an address
         6 (0.0%)      scanf("%s", name);
         1 (0.0%)      getchar();
         .
         6 (0.0%)      printf("Hello, %s!\n\n", name);
         6 (0.0%)  }
         .
         .
//////////////////////////////////////////////////////////////////////////
///////////////////////////////
         .
         4 (0.0%)  void exercise_2() {
         .             // Approximations of pi
         1 (0.0%)      int n=4;
         2 (0.0%)      double C=4.0; // Circumscribed semi perimeter
         2 (0.0%)      double I=2*sqrt(2); // Inscribed semi perimeter
         .
         2 (0.0%)      clock_t start = clock();
        13 (0.0%)      while (n < 100) {
        35 (0.0%)          C = (2 * C * I) / (C + I);
        35 (0.0%)          I = sqrt(C * I);
         5 (0.0%)          n *= 2;
         .             }
         2 (0.0%)      clock_t end = clock();
         7 (0.0%)      double time = ((double)(end - start)) /
```

```
CLOCKS_PER_SEC;
         .
         .              // Print approximate values
         6 (0.0%)       printf("n: %d\n", n);
         6 (0.0%)       printf("C: %f\n", C);
         6 (0.0%)       printf("I: %f\n", I);
         2 (0.0%)       printf("\n");
         .
         6 (0.0%)       printf("Elapsed %f seconds...\n\n", time);
         3 (0.0%)   }
         .
         .
///////////////////////////////////////////////////////////////////////////
///////////////////////////////
         .
         7 (0.0%)   void exercise_3() {
         6 (0.0%)       FILE* fp = fopen("mv.txt", "r");
         2 (0.0%)       if (fp == NULL) {
         .                  printf("ERROR reading file mv.txt...\n");
         .              } else {
         .                  int rows;
         .                  int cols;
         .                  // First line in file contains rows and columns
         7 (0.0%)           fscanf(fp, "%d %d", &rows, &cols);
         .                  // Now we can allocate the matrix
         8 (0.0%)           double* matrix = malloc(rows * cols *
sizeof(double));
         .                  // Vector length must be the number of columns
         6 (0.0%)           double* vector = malloc(cols * sizeof(double));
         .                  // Resulting product will be of length rows
         6 (0.0%)           double* result = malloc(rows * sizeof(double));
         .
         .                  // Read in the actual matrix
        45 (0.0%)           for (int i=0; i<rows; i++) {
       450 (0.0%)               for (int j=0; j<cols; j++) {
         .                          // Ignore white space
     1,500 (0.0%)                   fscanf(fp, "%lf", &matrix[i * cols +
j]);
         .                      }
         .                  }
         .
         .                  // Read in the vector length and ensure it is
cols
         .                  int vector_length;
         7 (0.0%)           fscanf(fp, "%d", &vector_length);
         4 (0.0%)           if (vector_length != cols) {
         .                      printf("ERROR - invalid dimensions given
with matrix of %d columns and vector of length %d", cols, vector_length);
         .                      // Prevent memory leak :-)
         .                      free(matrix);
         .                      free(vector);
         .                      free(result);
         .                      fclose(fp);
         .                      return;
         .                  }
         .
         .                  // Read in the vector
        45 (0.0%)           for (int i=0; i<vector_length; i++) {
       110 (0.0%)               fscanf(fp, "%lf", &vector[i]);
         .                  }
         .
         .                  // Perform multiplication
         2 (0.0%)           clock_t start = clock();
        45 (0.0%)           for (int r=0; r<rows; r++) {
        10 (0.0%)               int dot_prod = 0;
       450 (0.0%)               for (int c=0; c<cols; c++) {
     2,200 (0.0%)                   dot_prod += matrix[r*cols + c] *
vector[c];
         .                      }
        80 (0.0%)               result[r] = dot_prod;
         .                  }
         2 (0.0%)           clock_t end = clock();
         7 (0.0%)           double elapsed_time = ((double)(end - start)) /
CLOCKS_PER_SEC;
         .
         .                  // Print resulting vector
         3 (0.0%)           printf("Resulting Product:\n");
        45 (0.0%)           for (int i=0; i<rows; i++) {
       110 (0.0%)               printf("%f ", result[i]);
         .                  }
         6 (0.0%)           printf("\nTime Elapsed: %f seconds\n\n",
elapsed_time);
         .
         .                  // Prevent memory leak :-)
         3 (0.0%)           free(vector);
         3 (0.0%)           free(matrix);
         3 (0.0%)           free(result);
         3 (0.0%)           fclose(fp);
         .              }
         5 (0.0%)   }
         .
         .
//////////////////////////////////////////////////////////////////////////
///////////////////////////////
         .
         4 (0.0%)   void exercise_4(){
         .              // TODO - implement this function
         1 (0.0%)       int ITERATIONS = 10000000;
         .              volatile double result; // Because otherwise the
compiler is smart enough to realize we are not using the result and will
skip the code
         .
         .              // Time division
         2 (0.0%)       clock_t start = clock();
40,000,005 (2.3%)       for (int i=0; i<ITERATIONS; i++) {
50,000,000 (2.8%)           result = i / 1.0001;
         .              }
         2 (0.0%)       clock_t end = clock();
```

```
          7 (0.0%)      double total_operation_time = ((double)(end -
start)) / CLOCKS_PER_SEC;
          8 (0.0%)      printf("Division Time for %d Operations: %f
seconds\n", ITERATIONS, total_operation_time);
          .
          .             // Time multiplication
          2 (0.0%)      start = clock();
40,000,005 (2.3%)      for (int i = 0; i < ITERATIONS; i++)
          .             {
50,000,000 (2.8%)          result = i * 1.0001;
          .             }
          2 (0.0%)      end = clock();
          7 (0.0%)      total_operation_time = ((double)(end - start)) /
CLOCKS_PER_SEC;
          8 (0.0%)      printf("Multiplication Time for %d Operations: %f
seconds\n", ITERATIONS, total_operation_time);
          .
          .             // Time square root
          2 (0.0%)      start = clock();
40,000,005 (2.3%)      for (int i = 0; i < ITERATIONS; i++)
          .             {
70,000,000 (4.0%)          result = sqrt(i);
          .             }
          2 (0.0%)      end = clock();
          7 (0.0%)      total_operation_time = ((double)(end - start)) /
CLOCKS_PER_SEC;
          8 (0.0%)      printf("Square Root Time for %d Operations: %f
seconds\n", ITERATIONS, total_operation_time);
          .
          .             // Time sine
          2 (0.0%)      start = clock();
40,000,005 (2.3%)      for (int i = 0; i < ITERATIONS; i++)
          .             {
70,000,000 (4.0%)          result = sin(i);
          .             }
          2 (0.0%)      end = clock();
          7 (0.0%)      total_operation_time = ((double)(end - start)) /
CLOCKS_PER_SEC;
          8 (0.0%)      printf("Sine Time for %d Operations: %f
seconds\n\n", ITERATIONS, total_operation_time);
          3 (0.0%)  }
          .
          .
//////////////////////////////////////////////////////////////////////////
///////////////////////////////
          .
         10 (0.0%)  void exercise_5() {
          1 (0.0%)      int N = 128;
          .             volatile double v;
          .             clock_t start;
          .             clock_t end;
          .             double total_time;
         15 (0.0%)      while (N < 1024) {
          .                 // Static array
```

```
      3,561 (0.0%)           double static_array[N][N];
          .                  // Write to every cell to avoid memory cheating
messing with timing
      3,599 (0.0%)           for (int i = 0; i < N; i++)
          .                  {
  1,380,736 (0.1%)               for (int j = 0; j < N; j++)
          .                      {
  3,784,704 (0.2%)                   static_array[i][j] = 0;
          .                      }
          .                  }
          .
          .                  // Static array row major test
          6 (0.0%)           start = clock();
      3,599 (0.0%)           for (int i=0; i<N; i++) {
  1,380,736 (0.1%)               for (int j=0; j<N; j++) {
  3,784,704 (0.2%)                   v = static_array[i][j];
          .                      }
          .                  }
          6 (0.0%)           end = clock();
         21 (0.0%)           total_time = ((double)(end - start)) /
CLOCKS_PER_SEC;
         27 (0.0%)           printf("Static Array Size %d by %d Row Major
Time: %f Seconds\n", N, N, total_time);
          .
          .                  // Static array column major test
          6 (0.0%)           start = clock();
      3,599 (0.0%)           for (int i = 0; i < N; i++)
          .                  {
  1,380,736 (0.1%)               for (int j = 0; j < N; j++)
          .                      {
  3,784,704 (0.2%)                   v = static_array[j][i];
          .                      }
          .                  }
          6 (0.0%)           end = clock();
         21 (0.0%)           total_time = ((double)(end - start)) /
CLOCKS_PER_SEC;
         27 (0.0%)           printf("Static Array Size %d by %d Column Major
Time: %f Seconds\n", N, N, total_time);
          .
          .                  // Dynamic Array
         21 (0.0%)           double* dynamic_array =
malloc(N*N*sizeof(double));
          .                  // Write to every cell to avoid memory cheating
messing with timing
      3,599 (0.0%)           for (int i=0; i < N; i++) {
  1,380,736 (0.1%)               for (int j=0; j < N; j++) {
  3,784,704 (0.2%)                   dynamic_array[N*i+j] = 0;
          .                      }
          .                  }
          .
          .                  // Dynamic array row major test
          6 (0.0%)           start = clock();
      3,599 (0.0%)           for (int i = 0; i < N; i++)
          .                  {
```

```
 1,380,736 (0.1%)              for (int j = 0; j < N; j++)
         .                     {
 3,784,704 (0.2%)                  v = dynamic_array[i*N+j];
         .                     }
         .                 }
         6 (0.0%)          end = clock();
        21 (0.0%)          total_time = ((double)(end - start)) /
CLOCKS_PER_SEC;
        27 (0.0%)          printf("Dynamic Array Size %d by %d Row Major
Time: %f Seconds\n", N, N, total_time);
         .
         .                 // Dynamic array column major test
         6 (0.0%)          start = clock();
     3,599 (0.0%)          for (int i = 0; i < N; i++)
         .                 {
 1,380,736 (0.1%)              for (int j = 0; j < N; j++)
         .                     {
 3,784,704 (0.2%)                  v = dynamic_array[j * N + i];
         .                     }
         .                 }
         6 (0.0%)          end = clock();
        21 (0.0%)          total_time = ((double)(end - start)) /
CLOCKS_PER_SEC;
        27 (0.0%)          printf("Dynamic Array Size %d by %d Column
Major Time: %f Seconds\n\n", N, N, total_time);
         .
         9 (0.0%)          free(dynamic_array);
         6 (0.0%)          N *= 2;
         .             }
        10 (0.0%)  }
         .
         .
///////////////////////////////////////////////////////////////////////////
///////////////////////////////
         .
         .         // Helper function for exercise 6
         4 (0.0%)  void helper_exercise_6(char* c) {
         .             // Find the end of the string
         2 (0.0%)      char* start = c;
         2 (0.0%)      char* end = start;
        97 (0.0%)      while (*end != '\0' && *end != '\n') {
        11 (0.0%)          end++; // Pointer arithmetic
         .             }
         .             // Go back one from the terminating character if
the input was not empty
         3 (0.0%)      if (end > start) {
         1 (0.0%)          end--;
         .             }
         .             // Now we reverse the string
        19 (0.0%)      while (start < end) {
        15 (0.0%)          char temp = *start;
        20 (0.0%)          *start = *end;
        15 (0.0%)          *end = temp;
         5 (0.0%)          start++;
```

```
          5 (0.0%)          end--;
          .             }
          4 (0.0%)  }
          .
          .         // Transformer "Wrapper" Function
          6 (0.0%)  void wrapper_exercise_6(char* s, void
(*reverser_function_pointer)(char*)) {
          .             // Print original string
          6 (0.0%)      printf("Original String: %s\n", s);
          4 (0.0%)      reverser_function_pointer(s);
          6 (0.0%)      printf("Reverse String: %s\n\n", s);
          3 (0.0%)  }
          .
          4 (0.0%)  void exercise_6() {
          1 (0.0%)      int max_length = 100;
          5 (0.0%)      char* s = malloc(sizeof(char)*max_length);
          3 (0.0%)      printf("Enter a string to be reversed:\n");
          8 (0.0%)      if (fgets(s, max_length, stdin) != NULL)
          .             {
          6 (0.0%)          wrapper_exercise_6(s, helper_exercise_6);
          .             }
          .             else
          .             {
          .                 printf("Error reading input or EOF
reached.\n");
          .             }
          .
          3 (0.0%)      free(s);
          3 (0.0%)  }
          .
          .
/////////////////////////////////////////////////////////////////////////////
////////////////////////////////
          .
          6 (0.0%)  int main(int argc, char **argv) {
          2 (0.0%)      exercise_1();
          3 (0.0%)
printf("\n=============================================================\n"
);
          2 (0.0%)      exercise_2();
          3 (0.0%)
printf("\n=============================================================\n"
);
          2 (0.0%)      exercise_3();
          3 (0.0%)
printf("\n=============================================================\n"
);
          2 (0.0%)      exercise_4();
          3 (0.0%)
printf("\n=============================================================\n"
);
          2 (0.0%)      exercise_5();
          3 (0.0%)
printf("\n=============================================================\n"
);
         2 (0.0%)      exercise_6();
         .
         1 (0.0%)      return 0;
         2 (0.0%)

--------------------------------------------------------------------------
------
-- Annotation summary
--------------------------------------------------------------------------
------
Ir___________________

  431,023,854 (24.5%)    annotated: files known & above threshold &
readable, line numbers known
            0            annotated: files known & above threshold &
readable, line numbers unknown
            0          unannotated: files known & above threshold & two or
more non-identical
1,288,930,560 (73.2%)  unannotated: files known & above threshold &
unreadable
      395,882  (0.0%)  unannotated: files known & below threshold
   40,001,572  (2.3%)  unannotated: files unknown
```