CS 4473/5473: PDN Programming

Fall 2022

Dr. Richard M. Veras

C Refresher Part II

Goal: These problems are meant to get you into the groove of programming with C. In addition to providing correct code, you will also need to provide test cases that give you 100% coverage with **gcov** and 0 memory leaks reported under **valgrind**. The instructions for running gcov and valgrind will be provided at the end of the document. You may use outside code, but you must document its use. Your submission to canvas will be a tarball (submission.tar.gz) of the problems with the following directory structures:

./Problems

./Problems/Statistics/

./Problems/Transpose/

./Problems/Histogram/

Each directory will contain a prog.c, Makefile and the test cases that you use.

Problems: For each problem create the specified directory and include a **Makefile** and your created test files.

1. **./Problems/Statistics/prog.c**: On the command line read the name of the input file and the output file. The input file will start with an integer specifying the number of floating-point numbers that will follow. From that input file your program will compute the minimum value, maximum value, the average and the standard deviation. The results will be written to the output file specified earlier. Note: your standard deviation value might have significantly fewer digits than the example below. Hint: you should use malloc to create a buffer for the list of values.

./prog.x infilename outfilename	
Contents of **infilename**	Contents of **outfilename** after running prog.x
3 3.9 4.1 4.0	Min: 3.9 Max: 4.1 Avg: 4.0 Std: 0.081649658092772

2. **./Problems/Transpose/prog.c:** On the command line read the name of the input file and the output file. The first line of the input file will be two integers indicating the number of rows to follow and the number of columns in each row. The following lines represent values in a matrix (as floats). The output file contains the transposed values of the input file, starting with the number of rows and columns, followed by the values. Hint: you will need to dynamically allocate a 2d array.

./prog.x infilename outfilename	
Contents of **infilename**	Contents of **outfilename** after running prog.x
3 4 1 2 3 4 5 6 7 8 9 10 11 12	4 3 1 5 9 2 6 10 3 7 11 4 8 12

3. **./Problems/Histogram/prog.c:** On the command line read the name of the input file and the output file. The input file contains text and the output file is a histogram indicating the number of occurrences of each letter. Be mindful of the format used in the example.

./prog.x infilename outfilename	
Contents of **infilename**	Contents of **outfilename** after running prog.x
blaah	x xx x x _____ abcdefghijklmnopqrstuvwxyz

Hints: Everything you need to do this assignment can be found in these hints.

1. Makefiles explained: Here and Here
2. GDB is a powerful tool that is worth learning for these problems (run it using the terminal interface flag **gdb -tui**): Here, Here, and Here
3. How to use valgrind to find memory leaks: Here and Here
4. How to use gcov for code coverage: Here and Here (lcov is not necessary)
5. Pointers in C: Here, Here, and Here
6. Reading command line arguments in C: Here
7. Reading and writing a file in C: Here
8. Reading an integer in C: Here
9. Reading a float in C: Here
10. Using the math library in C: Here
11. Allocating an array in C: Here
12. Dynamically allocate a 2D array in C. Here

13. Creating a tarball: Here
14. Play with gcov and valgrind. Then write the Makefile. Below is an example.

example.c

```
/*
  This is a small program to demonstrate gcov. It
  can take no arguments or 1 argument. It prints
  a message if more than one is given. If the argument's
  value is odd it prints out odd, otherwise even.

  The goal is to show you need multiple test cases
  to cover all of the code.
  -richard.m.veras@ou.edu
*/

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
// No arguments passed
// note: argv[0] is the name of the
     // program, so we always have at least
     // 1 value in argv.
if (argc == 1+0)
     printf("I need more arguments!\n");
// If we have 1 argument
else if( argc == 1+1 )
{
// convert the argument to a value
int val = atoi(argv[1]);
if ( val % 2 == 0)
printf("Even!\n");
else
printf("Odd\n");
}
// more than 1 argument
else
printf("I need less arguments!\n");
return 0;
}
```

Makefile

```
# Note: the large spaces are tabs, this is important
# Run GCOV, requires that we build the code
# All of these runs of ./example are a test case
# We want to cover 100% of example through these tests
run-tests-coverage: build
```

```
./example
./example 3
./example 8
./example 5 4
gcov example.c

# Run Valgrind to find memory leaks, requires that we build the code
run-tests-leak: build
valgrind --tool=memcheck --leak-check=yes --show-reachable=yes --num-
callers=20 --track-fds=yes ./example
valgrind --tool=memcheck --leak-check=yes --show-reachable=yes --num-
callers=20 --track-fds=yes ./example 3
valgrind --tool=memcheck --leak-check=yes --show-reachable=yes --num-
callers=20 --track-fds=yes ./example 8
valgrind --tool=memcheck --leak-check=yes --show-reachable=yes --num-
callers=20 --track-fds=yes ./example 5 4

# Compile the code, requires that we clean up some files first.
# Add -lm if your code needs the math library.
build: clean
gcc -fprofile-arcs -ftest-coverage -g example.c -o example

clean:
rm -f *~
rm -f example
rm -f *.gcda *.gcno *.gcov
```