Midterm Exam

ECPE 170 – Computer Systems and Networks – Fall 2012

Name: Erich Viebrock

Tasks

Question #1:

Clone a Mercurial repository that contains a small program.

Document the command(s) used in your exam report.

Repository website: https://bitbucket.org/shafer/2012_fall_ecpe170_exam1

Answer:

```
cd ~/bitbucket/2012_fall_ecpe170
mkdir exam1
hg clone https://bitbucket.org/shafer/2012_fall_ecpe170_exam1
```

Question #2:

Copy the new program from the cloned repository into your personal BitBucket repository. Don't just leave the files cluttering up your repository. Rather, create a new folder called “exam1” at the same level as the existing “lab02”, “lab03”, … folders. Place the new program inside the “exam1” folder.

Document the command(s) used in your exam report.

Answer:

Before adding to BitBucket, I copied the files from the extracted repository folder, pasted them in the newly created “exam1” folder, then deleted the repository folder cloned from Mercurial, all via the GUI. I could have done this through the command line, but because it isn't one of the tasks, the GUI method was faster (for me, anyways).

```
hg add data.txt main.c main.h
hg status
hg commit -m "Completing the second question of the midterm exam; copying the source code files to my BitBucket repository."
hg push
```

Question #3:

Create a Makefile for the program. Your Makefile must enable the following options:

a. Compiler: GCC

b. C language standard: C99

c. Compiler optimization disabled (level “0”

d. Compiler provides warnings about all code that it considers questionable and potentially an error.

e. Output binary name: exam_program

Answer:

Initially I copied and pasted a Makefile from the post lab of Lab #5. Then, I edited the copy of that Makefile to meet the specifications of the new Makefile.

```
# The variable CC specifies which compiler will be used.
# (because different unix systems may use different compilers)
CC=gcc

# The variable CFLAGS specifies compiler options
#   -c :    Only compile (don't link)
#   -Wall:  Enable all warnings about lazy / dangerous C programming
#   -std=c99: Using newer C99 version of C programming language
CFLAGS=-c -Wall -std=c99

# All of the .h header files to use as dependencies
HEADERS=main.h

# All of the object files to produce as intermediary work
OBJECTS=main.o

# The final program to build
EXECUTABLE=exam_program

# ----------------------------------------

all: $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
     $(CC) $(OBJECTS) -o $(EXECUTABLE)

%.o: %.c $(HEADERS)
     $(CC) $(CFLAGS) -o $@ $<

clean:
     rm -rf *.o $(EXECUTABLE)
```

Question #4:

Compile the program and run it.

Document the command(s) used in your exam report.

Answer:

```
make clean
make
./exam_program
```

Question #5:

Measure the (very fast) execution time of the program in seconds.

Document the command(s) used in your exam report, and the results.

In your exam report, identify which one of the output numbers printed best represents "wall clock" time, as if you had measured it yourself with a stopwatch.

Answer:

```
make clean
make
time ./exam_program
real 0m0.019s
user 0m0.012s
sys  0m0.000s
```

"Real" most accurately represents real time.

Question #6:

The program has one memory related error. Use Valgrind to produce a report fie that tracks this problem down a specific line in a specific source code file. Depending on the type of error, Valgrind may report where a memory block was originally allocated. This is acceptable.

Document the command(s) used in your exam report, and include the full Valgrind report file.

Tip: Isn't there some compiler flag that must be set for Valgrind to understand your source code, and be able to refer to specific lines in specific files? Otherwise, it only knows about binary names and function names.

Answer:

```
make clean
make
valgrind --tool=memcheck --leak-check=yes --show-reachable=yes --num-
callers=20 --log-file=memcheck.txt ./exam_program
==2624== Memcheck, a memory error detector
```

```
==2624== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et
al.
==2624== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright
info
==2624== Command: ./exam_program
==2624== Parent PID: 1880
==2624==
==2624==
==2624== HEAP SUMMARY:
==2624==     in use at exit: 3,744 bytes in 468 blocks
==2624==   total heap usage: 1,611 allocs, 1,143 frees, 13,232 bytes
allocated
==2624==
==2624== 352 bytes in 44 blocks are indirectly lost in loss record 1
of 2
==2624==    at 0x402BE68: malloc (in
/usr/lib/valgrind/vgpreload_memcheck-x86-linux.so)
==2624==    by 0x80486C6: string_append (main.c:81)
==2624==    by 0x804860A: main (main.c:40)
==2624==
==2624== 3,744 (3,392 direct, 352 indirect) bytes in 424 blocks are
definitely lost in loss record 2 of 2
==2624==    at 0x402BE68: malloc (in
/usr/lib/valgrind/vgpreload_memcheck-x86-linux.so)
==2624==    by 0x80486C6: string_append (main.c:81)
==2624==    by 0x804860A: main (main.c:40)
==2624==
==2624== LEAK SUMMARY:
==2624==    definitely lost: 3,392 bytes in 424 blocks
==2624==    indirectly lost: 352 bytes in 44 blocks
==2624==      possibly lost: 0 bytes in 0 blocks
==2624==    still reachable: 0 bytes in 0 blocks
==2624==         suppressed: 0 bytes in 0 blocks
==2624==
==2624== For counts of detected and suppressed errors, rerun with: -v
==2624== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from
0)
```

Question #7:

Fix the memory error in the source code in a way that “preserves the intent of the programmer”. In other words, the program should still run, still produce the same output, and still exit normally. But, without the memory error.

Run Valgrind again to produce a clean report file.

Include the full Valgrind report file in your exam report.

Answer: