Lab Report

ECPE 170 – Computer Systems and Networks – Fall 2012

Name: Erich Viebrock

Lab Topic: C Programming (Lab # 3)

Lab Report

Question #1:
Look closely at the symbol table: What is the difference between the entries for "function_in_main" and "function_in_file2"? What does this actually mean?
Answer:
“Function_in_main” has a value of 3d, a size of 20, and is a function. “Function_in_file2“ has a value of 0, a size of 0, and does not have a type. “Function_in_main” also has an index of 1, while the index of “function_in_file2“ is not defined. This means whereas main was compiled and appears functional, file2 was not compiled in this object file and therefore appears empty.

Question #2:
How does "function_in_file2" differ in this object file from the previous object file? (What does this actually mean?)
Answer:
“Function_in_file2“ differs in this object file because it now has a size of 33, is a function, and has an index of 1. This means the object files compile relative to the .c file that is compiled via gcc.

Question #3:
Why doesn't "function_in_main" appear in this object file?
Answer:
“Function_in_main” doesn’t appear in this object file because it has not been linked to file2 yet.

Quesion #4:
What errors do you get when running this command? Explain why each of these error occur. (Hint: try man puts to see what that function does).
Answer:
When running this command, I got errors of undefined reference to “puts”, “function_in_file2”, “global_var_in_file2”, and “printf”. Function_in_file2 and global_var_in_file2 were errors because main.c was calling these functions, yet they are defined in a separate file apart from main.c. Because that separate file had not been linked, main.c had no way of processing the called functions. However, I'm stumped why puts and printf are errors when running this command. Puts and printf are used to output data and character strings to the GUI, while they only need the stdio library to be able to function. The stdio library header file is included in main.c, yet puts and printf are errors.

Question #5:
What errors do you get when running this command? Explain why each of these errors occur.
Answer:
Again, we are prompted with errors for puts and printf. It remains a mystery to me why these are errors, when the library is correctly included in the beginning of the program, which is the only necessity these commands need in order to be functional.

Question #6:
What happens when you try to run your linked program? Do you believe this error message? Why or why not?
Answer:
When I try to run the linked program, the terminal outputs “program” is not in my directory. However, when I display the files in my directory, “program” is clearly listed in bold green. Therefore, I do not believe the compiler.

Question #7:
Research online - what is in these mysterious files crt1.o, crti.o, crtbegin.o, crtend.o, and crtn.o?
Answer:
crtl.o – Couldn't find an answer for this one. Everyone online was complaining their linker wasn't working correctly because this file was missing. Couldn't find a direct answer as to what this file was specifically used for.
crti.o – Same as above, people get the error of “No such file or directory” when the linker looks for the file crti.o. Therefore I cannot specifically say what is in the file, other than it plays a role in the linking process.
crtbegin.o -
crtend.o -
crtn.o -

Question #8:
Copy and paste in your functional Makefile-1
Answer:

```
all:
	gcc main.c output.c factorial.c -o factorial_program
```

Question #9:
Copy and paste in your functional Makefile-2.
Answer:

```
all: factorial_program

factorial_program: main.o factorial.o output.o
	gcc main.o factorial.o output.o -o factorial_program

main.o: main.c
	gcc -c main.c

factorial.o: factorial.c
	gcc -c factorial.c

output.o: output.c
	gcc -c output.c

clean:
	rm -rf *.o factorial_program
```

Question #10:
Describe - **in detail** - what happens when the command "make -f Makefile-2" is entered. **How does make step through your Makefile to eventually produce the final result?**
Answer:
Make runs through the Makefile to handle the entire compiling precompiling process. Basically, all the steps needed just before actually running the program. This way, the user can simply just use the command ./program for whichever program they want to run, avoiding the steps to compile beforehand.

Question #11:
Copy and paste in your functional Makefile-3
Answer:

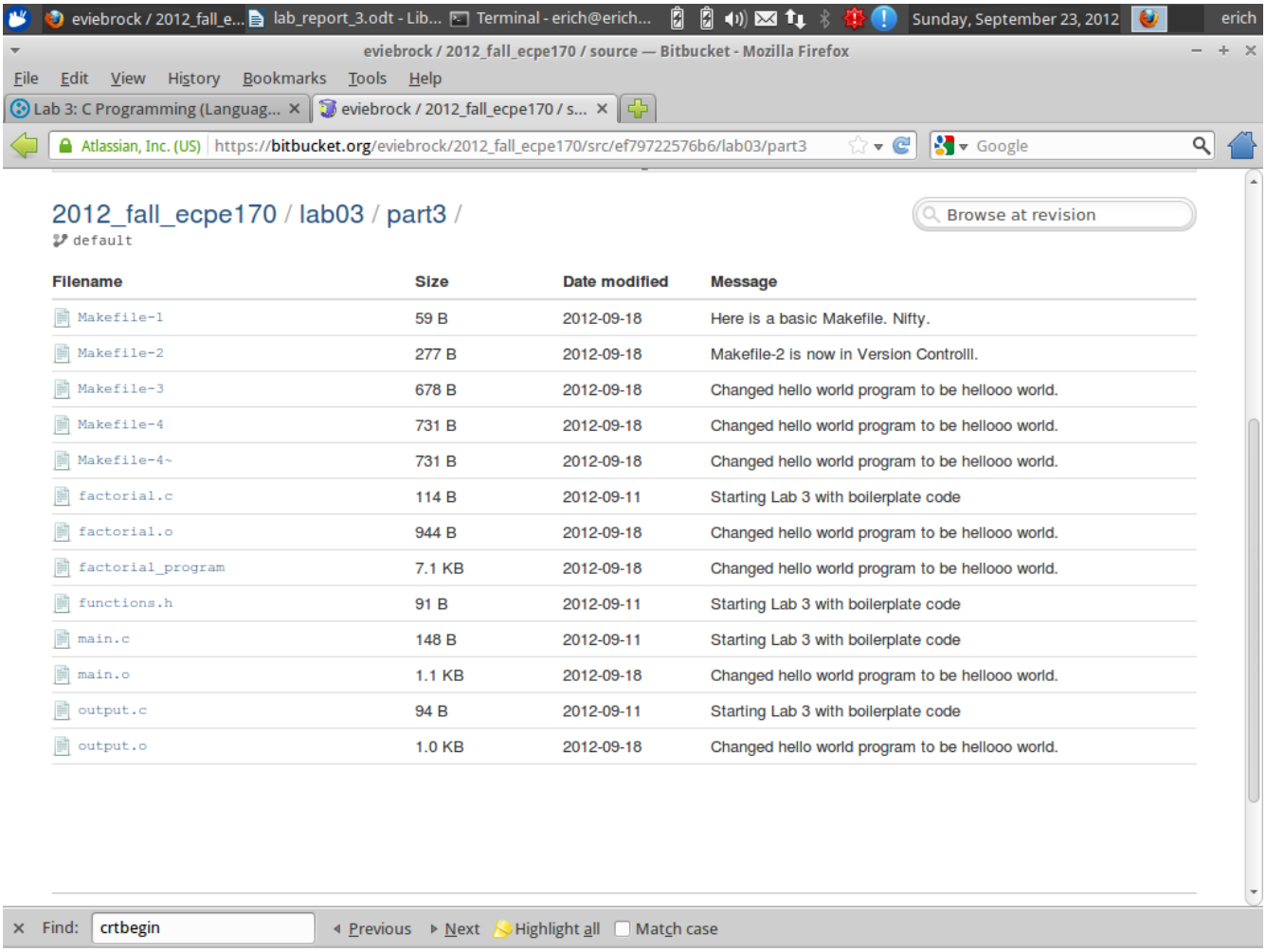
```
# The variable CC specifies which compiler will be used.
# (because different unix systems may use different compilers)
CC=gcc

# The variable CFLAGS specifies compiler options
#       -c :    Only compile (don't link)
#       -Wall:  Enable all warnings about lazy / dangerous C programming
CFLAGS=-c -Wall

# The final program to build
EXECUTABLE=factorial_program

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

all: $(EXECUTABLE)

$(EXECUTABLE): main.o factorial.o output.o
	$(CC) main.o factorial.o output.o -o $(EXECUTABLE)

main.o: main.c
	$(CC) $(CFLAGS) main.c

factorial.o: factorial.c
	$(CC) $(CFLAGS) factorial.c

output.o: output.c
	$(CC) $(CFLAGS) output.c

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

Question #12:
Copy and paste in your functional Makefile-4.
Answer:

```
# The variable CC specifies which compiler will be used.
# (because different unix systems ay 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
CFLAGS=-c -Wall

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

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

# The final program to build
EXECUTABLE=factorial_program

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

all: $(EXECUTABLE)

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

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

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

Question #13:
Explain what happens when make -f Makefile-4 is run
Answer:
Created all the object files and created the executable.

Question #14:
To use this Makefile in a future programming project (such as the post-lab assignment), what specific lines would you need to change?
Answer:



Question #15:
Take a screen capture of the Bitbucket.org website, clearly showing all of your Makefiles added to version control (along with the original boilerplate code)
Answer:

Question #16:
Why is it better to use hg cp instead of plain-old regular cp to copy a file within the repository?
Answer:
It is better to use “hg cp” instead of “cp”, because hg cp preserves version history.

Lab Report - Wrapup

Question #1
What was the best aspect of this lab?
Answer:
I found fully understanding the process behind compiling and linking very interesting.

Question #2
What was the worst aspect of this lab?
Answer:
I seem to have screwed up my ability to push anything to Mecurial, and trying to troubleshoot how to correct this issue was difficult.

Question #3
How would you suggest improving this lab in future semesters?
Answer:
If the different makefiles produced different output, it might make it easier to better understand makefiles and their processes.