Lab 1: Basic C Programming

Due: by the beginning of the next lab

1 Self-reproducing code (20 points)

In this problem, you will create a self-reproducing program or *quine*. Self-reproducing program, when executed, generates an output that is an exact copy of its own source code. For example, the following program is quine.

```
int main(){char *c="int main(){char *c=%c%s%c;printf(c,34,c,34);}";printf(c,34,c,34);}
```

To compile the program, first log in to *csce.unl.edu* or *CSCE* and type in the code above or copy and paste the code into an empty text file and name it *quine0.c*. Make sure that the file only has one long line and no blank lines. To compile the file, use the following command:

```
csce> gcc quine0.c -o quine0.
```

To run the program, use the following command:

```
csce> ./quine0
```

To ensure that this program produces a correctly working quine, use the following command to validate:

```
csce> ./quine0 > temp.out
csce> diff -w quine0.c temp.out
```

The `diff` command should not return any difference. No difference means that it is a correct quine.

Notice that when you compiled `quine0.c`, there is a warning from the compiler. The warning from CSCE would be as follows:

```
quine0.c: In function ‘main’: quine0.c:1:68: warning: incompatible implicit declaration
of built-in function ‘printf’ [enabled by default] int main(){char *c=‘‘int main()char
c=%c%s%c;printf(c,34,c,34);’’;printf(c,34,c,34);}
```

As your lab assignment, you need to remove this warning. To do so you have to add the following lines:

```
#include<stdio.h>

int main(){char *c="int main(){char *c=%c%s%c;printf(c,34,c,34);}";printf(c,34,c,34);}
```

Save the modified program as `quine1.c`. Note that you are asked to make three changes:

1. Add `#include<stdio.h>` and make it the first line in the file. Function `printf` is part of standard C library that perform I/O. Therefore, you should include `stdio.h` to avoid the warning.
2. Add a blank line right after it.
3. Add another blank line at the end of the program.

Compile the new program and name the binary `quine1`. The compiler should report no warnings. Try to run `quine1` and it should run fine. However, it is no longer a quine since the three changes are not included in the output. Your task is to modify the program so that it becomes a correctly working quine once again.

1.1 Hints

1. The lab leader will give a short presentation to explain the problem in greater details. Please pay attention to the lecture. The key in solving this problem is to understand that some special character such as " or " cannot be directly entered into `printf` statements. This is because " and " signify the beginning and end of a string to be printed. Other special characters such as % and "\n" also needs special treatments to have them appear correctly in the output. To overcome this issue, you'll need to directly enter ASCII values of these special characters. Visit `https://en.wikipedia.org/wiki/ASCII` or Chapter 2 of the course textbook for a table of ASCII characters.

2. Take the original program and try to have a better understanding how all the pieces fit together. For example, the last `printf` statement has the following arguments: `printf(c,34,c,34);`. When do we need to have "c" twice? Why do we need "34" twice? What does 34 represent? What do "%c%s%c" represent?

3. When you add a line (e.g., the blank line at the end of the code), make sure you have that working first. Do not to put everything in at once and then try to debug. Instead, add a change and then make sure that it works before moving on to another change.

4. If you get a segmentation fault error, it is most likely due to empty format placeholder (e.g., %d). Also make sure you also have the right place holder for the right type. For example, %s is for string, %c is for character, and %d for integer in decimal. A good reference C book by Kernighan and Ritchie (*The C Programming Language, Second Edition*) is publicly available on-line. You can find it via Google search. Please use it as a reference.

1.2 Grading

Redirect the output of your `quine1` program to a temporary file (e.g., temp1.out). Use `diff -w quine1.c temp1.out` to check for correctness of your quine. When `diff` reports no differences, your program is ready for grading. However, wait until you've completed the entire lab before asking a TA to check you off.

2 Memory Trace Processing (30 points)

We are providing you with two memory trace files *spice.din* and *cc.din*. They are available on canvas as part of *traces.zip*. Note that a trace file contains addresses issued by a CPU to execute an application. Your task is to write a C program to process the two trace files. In each file, there are about one million lines, and each line has two fields. The first field indicates what kind of operation the CPU performs on an address (read, write, initialize). The second field is the address in decimal. Note that the lines are also sorted from smallest address to largest address based on the second field. In this lab, you only need to consider the second field in the file. Your

task is to generate an output file that contains access frequency of each address. Here are the basic requirements for your program:

1. Your C executable program will be named `processTrace.c`. Since this is a C program, you should not have any C++ files. Files should be named with the extension .c, not .cpp. You cannot use C++ functions (e.g., cout, cin). Use C functions instead (e.g., fprintf, fscanf).

2. You must use linked list as the data structure to store information. Each node represents an address. There should also be a count field and a next field in each node. Refer to programs from the lectures to help you start.

3. Download the two trace files to the same directory as your program.

4. Use the following command to compile your program:
   `csce> gcc processTrace.c -o processTrace`

5. Use the following two commands to process the traces:
   `csce> ./processTrace cc.din cc.out` and `./processTrace spice.din spice.out`
   Note that *cc.out* and *spice.out* are the two output files.

6. After receiving a valid trace file, the program should process it and then generate an output file (either *spice.out* or *cc.out*) that has two fields separated by a space. For each line, the first field contains the **number of accesses to that address**, and the second field contains the **unsigned integer address**. The lines are sorted based on the address field from the smallest address to the largest address.

2.1 Hints

When in doubt, use the man page (e.g., man fscanf) to access detailed information regarding a particular function that you want to use.

Talk to the teaching assistant if you have questions about how to use the C input/output functions or have questions about the C language. Also refer to any good on-line C programming book (e.g., Kernigan and Ritchie's C Programming) for additional help.

2.2 Grading

We are providing two oracle files for you to use to compare your output files (available on canvas as *oracles.zip*). To check if you are producing the correct results, use `diff -w cc.out cc.ora` to check for correctness of your program. When `diff` reports no differences, your program is ready for grading. However, wait until you've completed the entire lab.

3 Having fun with C language (15 pts)

Download *lab*1*Problem*3*.zip* from *canvas* and then copy the downloaded file to a directory in your *csce.unl.edu* account. Once copied, unzip the file. Inside the *lab*1*Problem*3 directory, there are two programs: *prob*1*.c* and *prob*2*.c* and a shell script *run.sh*. To compile and run a program (e.g. *prob*1*.c*), you would issue the following commands:

- cd lab1Problem3 (You only need this command if you are not already in the lab1Problem3 directory.)
- chmod 755 run.sh
- ./run.sh (to compile the two programs)

The detailed descriptions of these two programs can be described as follows:

1. *prob1.c* is a simple C program that calls two functions $func1$ and $func2$. I want you to observe the output of this program. Notice that $result1$ is initially set to 2500, and $result2$ is set to -500. However, the code did not touch $result1$ after initializing it, but its value changes from 2500 to something else (e.g., -500 or 0). Your task is to explain why such a scenario occurs.
2. *prob2.c* is a short program to illustrate type casting. Your task is to explain how this process works by simply illustrating the memory content after myX is initialized and then do the same for the memory content of myY after casting.

3.1 Hints

To understand how these two programs work, you may need to use a debugger (GDB) or simply insert print statements to see how the actual address of a variable or how each byte is stored in a data structure. Some questions you may want to ask yourself include, for example, what are the return values of *function1* and *function2* and what is the size of type `long long int`?

For help with GDB, you can search Google for a good tutorial or use the following cheat sheet: `http://darkdust.net/files/GDB%20Cheat%20Sheet.pdf`.

3.2 Grading

Once you have thoroughly analyzed both programs, you are ready for a TA to check you off. Concisely explain to the TA what happens in these two programs to earn the 15 points. Specifically, the TA will ask the following questions:

1. What does the keyword `volatile` mean?
2. What type is `int * result1`?
3. What are the values stored in `result1` and `*result1` after the third line in `main` has been executed? Indicate the actual values in your program.
4. Who *incidentally* change the value of `* result1`?
5. What is the term used to describe this type of problem exhibited in *prob1.c* (choose one from the options below)?
   - garbage memory
   - dangling reference
   - segmentation fault

- bus error
- null pointer dereference

6. What is the value (in hex) stored in `f1` in *prob2.c*?
7. What are the byte values stored in `f2`? Please show each byte of the 18 bytes in `f2` in hex.
8. What are the sizes in byte of `struct x` and `struct y`? Hint: modify the program to use `sizeof` operator.

4 Submission

Submit only the two source files (`quine1.c` and `processTrace.c`) via canvas once you have been checked off by a TA.