CSE 374 Midterm Exam 5/2/22 Sample Solution

Question 1. (12 points, 3 each) Linux commands and shell expansion.

Suppose that the current directory contains the following files

```
foo.c       cow.story     cow.cat       cow.cc
tickle.c    dog.cow       story.txt     cow.h
```

What output is produced when each of the following bash commands is executed in this directory?

(a) echo *c

`cow.cc foo.c tickle.c`

(b) echo *.c

`foo.c tickle.c`

(c) echo *.c*

`cow.cat cow.cc dog.cow foo.c tickle.c`

(d) echo *.*.c

`*.*.c`

Grading note: for part (d), `echo` prints the original pattern because no files match `*.*.c`. Answers that indicated no match or were blank also received full credit.

Question 2. (6 points) Aliases. To compile the code for hw4, we used this command:

```
gcc -Wall -g -std=c17 -o hw4 hw4.c
```

That gets tediously very quickly. We'd like to save some typing by defining an alias `compile` that allows us to do the same thing but without having to type as much.

Define a bash alias `compile` so that typing

```
compile -o hw4 hw4.c
```

does exactly the same thing as the original `gcc` command given above. (Of course, different arguments could be given after `compile`, not just `-o hw4 hw4.c`, so the alias should execute `gcc` with the required options and whatever other arguments are present.)

(Hint: the answer is quite short – you will not need nearly this much space to write it)

```
alias compile="gcc -Wall -g -std=c17"
```

Note: the quotes are required, otherwise the alias would not properly include the options and the whitespace separating them. Single quotes would also work.

Question 3. (15 points) (A little scripting) The alias from the previous problem is better than having to type an entire `gcc` command, but we could do even better with a shell script. For this problem write a bash script that compiles programs with `gcc` as follows: If the script is stored in an executable file named `comp`, then the command

```
./comp foo
```

should execute the command `gcc -Wall -g -std=c17 -o foo foo.c`. If there are two arguments naming the source file and compiled file, in that order, then

```
./comp bar.c foo
```

should execute the command `gcc -Wall -g -std=c17 -o foo bar.c`. If the script does not have exactly one or two arguments, then it should print a suitable error message and quit with an `exit 1`. If the script executes a `gcc` command without printing an error message, then it should exit with status 0.

Write your answer below. The `#!/bin/bash` first line of the script is provided for you.

```
#!/bin/bash

if [ $# -eq 1 ]
then
   gcc -Wall -g -std=c17 -o $1 $1.c
   exit 0
fi

if [ $# -eq 2 ]
then
   gcc -Wall -g -std=c17 -o $2 $1
   exit 0
fi

# wrong # arguments
echo "$0: must have 1 or 2 arguments"
exit 1
```

There are, of course, other correct answers to this problem, and any correct answer received credit. The error message printed did not need to include $0 (the command name) and the wording could be anything appropriate.

Question 4. (12 points, 2 each) grep. Suppose we have a file hello.txt containing this text (including the // line numbers at the end of each line):

```
Hello World          // Line 1
hello world          // Line 2
world hello          // Line 3
WORLD HELLO          // Line 4
hello hello world    // Line 5
hello world world    // Line 6
```

For each of the following `grep` commands, circle the numbers of the lines from this file that will appear in the `grep` output. Note that `grep -i` compares strings ignoring case (i.e., `grep -i Hello` … will match `Hello`, `hello`, `heLLo`, and so forth). Hint: recall that `[aw-z]` matches a, w, x, y, z; and `[^aw-z]` matches all *other* characters besides a, w, x, y, z. A single ^ in a pattern outside of [ ] brackets means something else.

(a) `grep "hello world" hello.txt`

1 (2) 3 4 (5) (6)

(b) `grep -i "^hello world" hello.txt`

(1) (2) 3 4 5 (6)

(c) `grep "[^hello]" hello.txt`

(1) (2) (3) (4) (5) (6)

(d) `grep "^[^hello]" hello.txt`

(1) 2 (3) (4) 5 6

(e) `grep "world$" hello.txt`

1 2 3 4 5 6

(f) `grep "world.*$" hello.txt`

1 (2) (3) 4 (5) (6)

Unfortunately, this question misfired. The intent was to indicate which lines in the file matched each regular expression. The way it was worded, however, many people read it as "circle the number of matches" (which can't be quite right since 0 is not one of the possible answers). When we graded the problem we allowed almost full credit for answers that read the question that way and gave the correct total number of matches for each part.

Question 5. (14 points, 7 each) (`sed`) We have a file `snake.txt` containing the following:

```
    Sydney the snake slid down a slippery slope.
 Since getting stuck in a snare made of sticks and some string, she'd
        somehow succumbed to an embarrassing, sibilant stutter. She was sneakily
in search of a slimy, secret spot where she could sink into obscurity.
        Sadly, she never found it.
```

(credit: Sydney the Stuttering Snake, adapted from http://fiftywordstories.com/tag/alliteration/)

For each of the following, use `sed` with a single '`s`' command and basic regular expressions (not extended) to produce the requested results. You may also include a `grep` command if needed. The solution must be a single-line command that could be typed in a bash shell window, assuming that `snake.txt` is in the current directory.

(a) Write a command to remove all leading whitespace (spaces and tabs) from `snake.txt`, and store the answer in file `snake2.txt`. (Hint: use `\t` to include a tab in a regular expression)

```
sed -e 's/^[ \t]*//g' snake.txt > snake2.txt
```

(Grading note: it turns out that using `\t` for tab is implemented in the Linux version of `sed`, but is not part of the official specification for `sed` and doesn't actually work on some other common versions, including macOS, amazingly enough. But since it works for Linux and because `\t` is the normal escape for tab, we used it here.)

(b) Write a command to replace all double pairs of letters with a single copy of that letter in `snake.txt`. For example, the word "getting" should be replaced with "geting" and "succumbed" replaced with "sucumbed". Write the results to standard output.

```
sed -e 's/\(.\)\1/\1/g' snake.txt
```

There are, of course, other possible solutions to both of these problems. However, a common error was to use `grep` to select lines matching some or all of the regular expression, then use `sed` to edit the selected lines. That omits lines from the output that did not match the original `grep` command, even though they should have been included in the result.

Question 6. (18 points) The traditional, annoying C program. As is usual, this program compiles and executes without warnings or errors, although this time it might contain a bug or two that causes unexpected output, but without a crash.

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

void copyString(char* src, char* dst, int len) {
  for (int i = 0; i < len; i++) {
    dst[i] = src[i];
  }
  // Draw picture for part (a) when execution gets HERE
  // the 2nd time copyString is called
}

int main() {
  char* hello = "Hello";
  char* names[] = {"Bennedict", "Xinyue"};
  char name[10] = {0}; // initializes all chars in name to \0
  copyString(names[0], name, strlen(names[0]));
  printf("%d: %s %s\n", 0, hello, name);
  copyString(names[1], name, strlen(names[1]));
  printf("%d: %s %s\n", 1, hello, name);
  return EXIT_SUCCESS;
}
```

(a) (12 points) On the next page, draw a diagram showing the contents of memory when execution reaches the "draw picture" comment at the end of function `copyString`. the *second* time it is called. Be sure to show boxes for the local variables and parameters of all active functions, including `main`. If a variable is a pointer, indicate its value by drawing an arrow between the variable and the storage location (variable) that it points to. Hint: you will probably find it convenient to trace execution and draw the diagram from the beginning and stop when you reach the end of `copyString` the second time. Be sure to clear up any old values of pointers or variables so the diagram clearly shows memory when we reach the end of `copyString` the second time.

(b) (6 points) What output is produced by this program when it is executed?

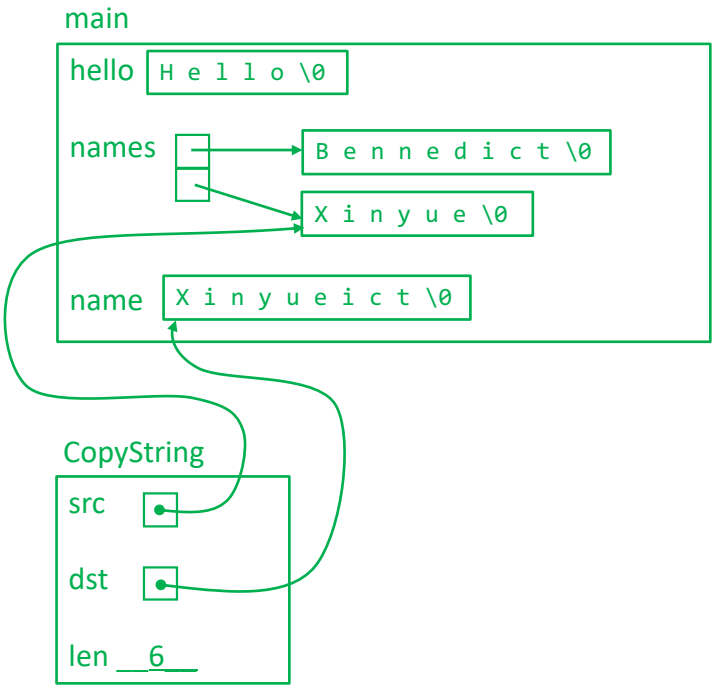
```
0: Hello Bennedict
1: Hello Xinyueict
```

(write your answer to part (a) on the next page)

Question 6. (cont) (a) Draw your diagram below showing the contents of memory when execution reaches the "draw picture" comment in function `copyString` the *second* time. Code repeated for convenience.

```
void copyString(char* src, char* dst, int len) {
  for (int i = 0; i < len; i++) {
    dst[i] = src[i];
  }
  // Draw part (a) picture when execution gets HERE the 2nd time
}

int main() {
  char* hello = "Hello";
  char* names[] = {"Bennedict", "Xinyue"};
  char name[10] = {0}; // initializes all chars in name to \0
  copyString(names[0], name, strlen(names[0]));
  printf("%d: %s %s\n", 0, hello, name);
  copyString(names[1], name, strlen(names[1]));
  printf("%d: %s %s\n", 1, hello, name);
  return EXIT_SUCCESS;
}
```



A common error here was omitting the \0s that mark the end of each C string.

Another issue is that it's not entirely clear from what we've covered exactly how to diagram the `names` array. We allowed a lot of flexibility in answers. As long as the diagram indicated that it contained constant string values with terminating \0 characters, and not a separate variable or data structure allocated on the heap or somewhere else, the answer received credit.

Question 7. (20 points) The small C programming exercise. Write a small C program that prints the number of times its first argument string appears in the all of the command-line arguments when it is executed. If the compiled program is named `countargs`, then execution should produce the following output:

```
./countargs foo
foo appears once
./countargs foo foo
foo appears 2 times
./countargs foo bar foo foo baz fi
foo appears 3 times
```

If the program does not have at least one command-line argument (`foo` in the above example), it should print an appropriate error message and exit with an appropriate status code. You should not include the program name itself (`./countargs` above) in the count, even if the first argument is the same as the program name. Just count how many times the first argument appears starting with the first argument.

The usual page of reference information is included at the end of the exam, although you may not need most of that information since this program only examines the strings in the argument list and does not read or write any files (other than messages to `stdout`).

Write your answer below. Some #inclues that you might find useful are written for you. You can include all of the code in a single `main` function if you wish.

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

int main(int argc, char **argv) {
  int ntimes;  // number of times argv[1] appears

  if (argc < 2) {
    printf("%s requres at least one command line argument\n",
                argv[0]);
    exit(EXIT_FAILURE);
  }

  ntimes = 1;
  for (int i = 2; i < argc; i++) {
    if (strcmp(argv[1], argv[i]) == 0) {
      ntimes++;
    }
  }

  if (ntimes == 1) {
    printf("%s appears once\n", argv[1]);
  } else {
    printf("%s appears %d times\n", argv[1], ntimes);
  }

  return EXIT_SUCCESS;
}
```

Again, there are many possible solutions. Any correct program received credit.

Question 8. (3 *free* points) (All reasonable answers receive the points. All answers are reasonable as long as there is an answer. ☺)

(a) (2 point) What question were you expecting to appear on this exam that wasn't included?

Many answers here….

(b) (1 points) Should we include that question on the final exam? (circle or fill in)

Yes

No

Heck No!!

$!@$^*% No !!!!!

Yes, yes, it *must* be included!!!

No opinion / don't care

None of the above. My answer is _____.