CSE 374 Midterm Exam 5/2/22

Name _____ UW Id # _____
(please print legibly)

There are 8 questions worth a total of 100 points. Please budget your time so you get to all of the questions. Keep your answers brief and to the point.

The exam is closed book, closed notes, closed electronics, closed telepathy, etc., except that you may have one 5x8 card with any hand-written notes you wish on both sides.

Many of the questions have short solutions, even if the question is somewhat long. Don't be alarmed.

If you don't remember the exact syntax of some command or the format of a command's output, make the best attempt you can. We will make allowances when grading.

There is a blank page at the end with extra space for answers if you need more room.

The **last page** of the exam contains reference information that may be useful while answering some of the questions. Do not write on this page – it will not be examined while grading. You should remove this page from the exam and return it for recycling when you are done.

Relax, you are here to learn.

Please wait to turn the page until everyone is told to begin.

Score _____ / 100

1. _____ / 12
2. _____ / 6
3. _____ / 15
4. _____ / 12
5. _____ / 14
6. _____ / 18
7. _____ / 20
8. _____ / 3

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

(b) echo *.c

(c) echo *.c*

(d) echo *.*.c

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)

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
```

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

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)

(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.

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?

(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;
}
```

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>
```

(additional space for your answer on the next page)

Question 7. (cont.) Additional space for your answer if needed.

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?

(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 _____.

Additional space for answers if needed. Please indicate clearly which questions you are answering here, and also be sure to indicate on the original page that the rest of the answer can be found here.

Reference Information

Some of this information might be useful while answering questions on the exam. Feel free to remove this page for reference while you work. Please do not write on this page – anything written here will not be graded.

Shell: Some of the tests that can appear in a [ ] or [[ ]] test command in a bash script:

- string comparisons: =, `!=`
- numeric comparisons: `-eq`, `-ne`, `-gt`, `-ge`, `-lt`, `-le`
- `-d` *name* test for directory
- `-f` *name* test for regular file

Shell variables: `$#` (# arguments), `$?` (last command result), `$@`, `$*` (all arguments), `$0`, `$1`, … (specific arguments), `shift` (discard first argument)

Strings and characters (`<string.h>`, `<ctype.h>`)

Some of the string library functions:

- `char* strncpy`(*dest*, *src*, *n*), copies exactly *n* characters from *src* to *dst*, adding '\0's at end if fewer than *n* characters in *src* so that *n* chars. are copied.
- `char* strcpy`(*dest*, *src*)
- `char* strncat`(*dest*, *src, n*), append up to *n* characters from *src* to the end of *dest*, put '\0' at end, either copy from *src* or added if no '\0' in copied part of *src*.
- `char* strcat`(*dest*, *src*)
- `int strncmp`(*string1*, *string2*, *n*), <0, =0, >0 if compare <, =, >
- `int strcmp`(*string1*, *string2*)
- `char* strstr`(*string, search_string*)
- `int strnlen`(*s, max_length*)
- `int strlen`(*s*)
- Character tests: `isupper`(*c*), `islower`(*c*), `isdigit`(*c*), `isspace`(*c*)
- Character conversions: `toupper`(*c*), `tolower`(*c*)

Files (`<stdio.h>`)

Some file functions and information:

- Default streams: `stdin`, `stdout`, and `stderr`.
- `FILE* fopen`(*filename*, *mode*), modes include "r" and "w"
- `char* fgets`(*line, max_length, file*), returns `NULL` on end of file
- `int feof`(*file*), returns non-zero if end of *file* has been reached
- `int fputs`(*line, file*)
- `int fclose`(*file*)

A few `printf` format codes: `%d` (integer), `%c` (char), `%s` (char*)