CSE 374 Midterm Exam 11/1/17

Name _____ Id # _____

There are 6 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.

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.

Relax, you are here to learn.

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

Score _____ / 100

1. _____ / 8

2. _____ / 8

3. _____ / 20

4. _____ / 16

5. _____ / 24

6. _____ / 24

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 may remove that page from the exam if you wish.

Question 1. (8 points, 2 each) Linux commands. Here is a brief transcript from a Linux terminal session (user input follows each `$` prompt, the rest is system output):

```
$ cd
$ pwd
/home/user
$ ls -l
drwxr-xr-x 2 user uw_ugrad  4 Oct 28 11:33 start
$ mkdir finish
$ ls start
atextfile  btextfile  adir  dtextfile
$ cd start
$ ls adir
catextfile  cbtextfile
```

After each of the following commands, write the output that it produces. You should assume that the first command (`pwd`) is executed immediately after the above commands and each subsequent line of commands is executed after the commands in the previous parts of the question have been executed (i.e., the commands in parts (a) through (d) are run one after the other).

(a) `pwd`

(b) `mv adir/* ../finish/. ; ls`

(c) `echo $HOME`

(d) `cd ..; ls -l finish | wc -l`

Question 2. (8 points, 4 each) Aliases. Give `alias` commands that will create aliases that work as described below.

(a) Define an alias `datedir` that will print the current date, the path to the current directory, and the contents of that directory listed with details including the file dates and permissions. Example:

```
$ datedir
Tue Oct 30 15:43:59 PDT 2017
/home/user/docs
total 12
-rw-rw-r-- 1 user user 68 Oct 30 12:31 hello.c
-rw-rw-r-- 1 user user 17 Oct 28 15:43 README
-rw-rw-r-- 1 user user 35 Oct 12 13:15 story.txt
```

(b) The `rm` command provides an option `-i` that will ask the user to confirm that the files should actually be deleted before attempting to remove them. Define an alias `delete` that will execute `rm -i` on the file or files given as arguments to `delete`. Example (the "y" answers are provided by the user, they are not printed by the alias or by `rm`):

```
$ delete story.txt README
rm: remove regular file ‘story.txt’? y
rm: remove regular file ‘README’? y
```

Question 3. (20 points) (A little scripting) Anticipating winter break, you've accumulated a collection of books on your computer in plain ascii text files. Someone asked you how many words are in a typical book, and, being an experienced Linux hacker, you've decided to write a shell script to find out.

Write a script that has one or more file names as arguments and prints to `stdout` the average number of words in the files. In other words, the script needs to figure out the total number of words in the files, divide by the number of files, and print the result.

If no arguments are provided, print an appropriate error message to `stderr` (stream 2) and exit with return code 1. Otherwise you can assume that the arguments are one or more file names and you should calculate and print the average length of the file(s) to `stdout` and then exit with return code 0. Your script should handle file names that have embedded blanks in them like "short story". Hints: wc -w (to calculate number of words); integer arithmetic is good enough – don't worry about accuracy beyond that.

Write your answer below. The `#!/bin/bash` that starts the script is provided for you.

```
#!/bin/bash
```

Question 4. (16 points) (`sed`) One of our colleagues has been doing some data extraction and has produced a file `urls.txt` that contains a collection of hyperlinks. The input file has this format:

```
CSE: <a href="http://www.cs.washington.edu">UW Allen School</a> http link
Google uses https <a href="https://google.com">Google</a>
<a href="http://www.gnu.org/software/sed/manual/sed.html">sed manual</a>
```

Notice that some URLs begin with `http` while others start with `https`. Some of the descriptions like "UW Allen School" have embedded spaces, others don't. Some of the lines have text before the beginning "<a", others don't. Some have text following the "" at the end of the link, others don't. You can assume that none of the text before or after the hyperlink has additional "<a …>" or "" tags in it. You can also assume there is no extra whitespace or additional information besides the "href=" information inside the "<a …>" tags. In other words, the input will be like the example without additional pathological cases to consider.

Give a single `sed` command that will read this `urls.txt` file and write to `stdout` the description in each link (the text between `<a …>` and `</a>`), followed by a colon and a space, and then followed by the URL from the link. For the above input, the output of your `sed` command should be:

```
UW Allen School: www.cs.washington.edu
Google: google.com
sed manual: www.gnu.org/software/sed/manual/sed.html
```

Write your `sed` command below:

Question 5. (24 points) The traditional, annoying C program. As is usual, this program compiles and executes without warnings or errors.

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

void mystery(int* p, int* q, int n) {
  *q = p[1];
  *p = n+1;
  n = 44;
  /* draw picture at this point for part (a) */
  printf("during: %d %d %d\n", *p, *q, n);
}
void question(int* x, int *y) {
  int a;
  a = 33;
  printf("before: %d %d %d\n", *x, x[1], a);
  mystery(x, &a, *y);
  printf("after: %d %d %d\n", *x, x[1], a);
}
int main(int argc, char* argv[]) {
  int p[2];
  int n = 0;
  p[0] = 11;
  p[1] = 22;
  question(p, &n);
  printf("done: %d %d %d\n", p[0], p[1], n);
  return 0;
}
```

(a) (12 points) On the next page, draw a diagram showing the contents of memory when execution reaches the "draw picture" comment in function `mystery`. Be sure to show boxes for the local variables and parameters of all active functions. 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.

(b) (12 points) Fill in the lines below to show the output produced when this program is executed.

`before:` _____ _____ _____

`during:` _____ _____ _____

`after:` _____ _____ _____

`done:` _____ _____ _____

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

Question 5. (a) Draw your diagram here showing the contents of memory when execution reaches the "draw picture" comment in function `mystery`.

Question 6. (24 points) The small C programming exercise. A *palindrome* is a string that contains the same sequence of characters forward or backward. For instance, "mom", "abba" and "racecar" are palindromes. For this problem, write a C program that will read a text file and print to `stdout` each line from the file that is a palindrome.

Simplification: For this problem, a palindrome is a string that is absolutely identical in both directions. So, for example, "racecar" is a palindrome, but "Racecar" is not because the "R" at the front is not the same character as the "r" at the end. If there are spaces or punctuation characters in the string they should be treated the same as any other character – you do not need to do anything special to handle them. So "race car" is not considered to be a palindrome for this question, while "no on" would be.

You should use standard C library functions in your solution when appropriate (the last page of the exam contains a summary of some that might be useful). You do not need to write `#include` directives – assume this has been done for you.

(a) (8 points) Complete the function `is_palindrome(s)` below so it returns 1 (true) if its string parameter is a palindrome and 0 (false) if it is not. The string `s` is a proper C string – an array of characters with a '\0' byte at the end. The terminating '\0' byte is not part of the data that should be tested to see if the string is a palindrome – it simply marks the end of the string.

```
// return 1 if s is a palindrome or 0 if it is not
int is_palindrome(char *s) {




}
```

(continued on next page)

Question 6. (cont.) (b) (16 points) Now, write the main program that opens the file given as an argument to the program (i.e., `argv[1]`), read each line in that file, and, if the characters in that line forms a palindrome, print that line to `stdout`. If the file name is missing, or if the file cannot be opened, the program should print a suitable message to `stderr` and terminate with `EXIT_FAILURE`. If no error is detected, the program should terminate with `EXIT_SUCCESS` when it is done. Your program must use `is_palindrome(s)` from part(a) to decide if individual input lines are palindromes.

You may assume that no input line has more than 100 characters, including any newline ('\n') or '\0' bytes at the end of each line. An appropriate name has been `#define`d for you to use. You may also assume that there is a '\n' newline at the end of the last line in the file if it matters. Hints: be sure that any '\n' characters at the end of input lines are not accidentally included when testing whether a line is a palindrome. You should not use dynamic allocation (`malloc`/`free`) in your code – it is not needed.

```
#define MAX_LINE_LENGTH 100

int main(int argc, char **argv) {









}
```

(More space is provided on the next page for the rest of your answer if needed.)

Question 6. (cont.) Extra space for your answer if needed.

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*)