University of Toronto
Faculty of Arts and Science

August 2017 Examinations

CSC 209H1Y

Duration: 3 hours

Aids allowed: Any books and papers.
No electronic aids allowed: No calculators, cell phones, computers, ...
To pass the course you must achieve at least 35% on this exam.

Seat: BA2145 seat 1

Logname: chanalv4

Student: Chan, Alvin

Make sure you have all 12 pages (including this page).
(Don't panic about the page count—there's lots of space for answers.)

Answer *all* questions. Answer questions in the space provided. Answers not in the correct space will not be graded unless a note in the correct space says "see page ..." and the answer on that page is clearly labelled with the question number.

Be careful not to get stuck on some questions to the complete exclusion of others. The amount of marks or answer-space allotted does not indicate how long it will take you to complete the question, nor does the size of the answer-space indicate the size of the correct answer.

In general, in the C programming questions you can omit the #includes, and you can omit comments unless you need to explain something unclear about the functioning of your code.

Do not open this booklet until you are instructed to.

Do not write anything in the following table:

question	value	grade	question	value	grade
1	8		6	15	
2	6		7	15	
3	4		8	10	
4	12		9	8	
5	10		10	12	
subtotal			total	100	

1. [8 marks]
Please be careful to write single quotes and backquotes correctly to avoid misinterpretation.

a) Write a shell script which reads zero or more integers from the standard input, one per line, and outputs their sum. You can assume that the input is correctly-formed.
Examples, if the script is in a file named "add", where the '$' is the shell prompt:

```
$ (echo 2; echo 3; echo 6) | sh add
11
$ echo 2 | sh add
2
$ sh add </dev/null
0
$
```

b) Write a shell script which takes zero or more integers as command-line arguments and outputs their sum. You can assume that all command-line arguments are valid integers.
Examples:

```
$ sh add 2 3 6
11
$ sh add 2
2
$ sh add
0
$
```


continued...

2. [6 marks]
Write a complete shell script which behaves as follows. It must be run with at least one argument. Arguments are considered to be plain files. Appropriate messages should be printed for files which can't be read (by using unix tools appropriately so as to generate messages).

The output of your shell script is the size in bytes of the largest of the files, with the file name. If there is a tie for largest, output any one of the files.

Example session, if your script is in a file named "largest":

```
$ sh largest file1 file2 file3
14680 file1
$
```

3. [4 marks]
How many groups are you in? The output of the "id" command looks like this (all on one line):

```
uid=123(leephy98) gid=1008(cstudent) groups=1008(cstudent),517(csc209h),
525(csc263h)
```

Since the "gid" value is always repeated in the "groups=" list, and since you are necessarily in at least one group, all you need to do to answer the question "how many groups are you in?" is to count the commas in the "id" output and add one.

Write shell commands to output the number of groups you are in.

continued...

4. [12 marks]
There are three different times in an inode in the unix filesystem: the mtime (last-modified time), the atime (last-accessed time), and the ctime (last inode change time). Suppose that there is a plain file named "abc" in the current directory:

a) Write a shell command which will update the mtime of "abc" (i.e. set it to the current time).

b) Write a shell command which will update the atime of "abc", but not change its mtime or ctime.

c) Write a shell command which will update the ctime of "abc", but not change its mtime or atime.

d) Write a shell command which will update the mtime of the current directory.

e) Write a shell command which will update the atime of the current directory.

f) Write a shell command which will update the ctime of the current directory.

continued...

5. [10 marks]
Write a partial version of the *test* command in C. Your program will implement only the following features of *test*:

```
test -f file
test -d file
test -s file
```

"test –f file" tests whether the file exists and is a plain file.
"test –d file" tests whether the file exists and is a directory.
"test –s file" tests whether the file exists and is a plain file AND is non-zero-sized.

continued...

6. [15 marks]
Write a modified and partial version of the *tr* command in C. Your program requires at least two arguments (i.e. argc≥3), where invocations will look like this:

```
tr e f
tr e f file1 file2
tr 0123456789 X file
```

The first argument is a string of one or more characters to translate. All of these characters are translated to the single character which is the second argument. For this exam question, you do not need to check that argv[2] is of length 1 (although you do need to check that it exists, by checking argc).

After these two mandatory arguments, your program takes zero or more files in the usual way, where zero filename arguments means to read the standard input. As always, the output is to the standard output.

You may want to use "strchr()" to check whether a character occurs in a string.

continued...

(more space for your question 6 answer, if required)

continued...

7. [15 marks]
Write a C program which recursively traverses the entire filesystem, starting from the root directory. It adds together the sizes, in bytes, of all files in the filesystem, including special files (including directories), and outputs this number. (After a successful stat() or lstat() call, the size of the file in bytes is present in the struct member "st_size".) You can assume that the total can be represented in an int.

If a complete file path name reaches 1000 characters or more in length, your program may terminate with an appropriate error message; but it does have to check, and it may not exceed array bounds.

continued...

8. [10 marks]
There is a computation which takes a long time to run, but is broken up into five roughly equal parts which can run simultaneously.

Write a function in C (not a complete program; no main()) which forks five times; each of the child processes runs compute(n) for a different $0 \leq n < 5$; and your function wait()s for all five child processes before returning.

(There's no exec(); all of this happens within one program.)

You don't have to keep track of process ID numbers; you can assume for this question that there are no other child processes, so a wait() call will always wait for one of the five processes you have spawned.

9. [8 marks]
The program /local/t3 writes some data to file descriptor 7. Write a complete C program which execs /local/t3 with its file descriptor 7 redirected to the file "bar" (in the current directory) (the standard file descriptors 0, 1, and 2 are left alone). (All appropriate error checking is required.)

continued...

10. [12 marks]
The following program is a server which reads one byte (character) from each client, echoes it back, and drops the connection.

```
 1  #include <stdio.h>
 2  #include <unistd.h>
 3  #include <string.h>
 4  #include <sys/types.h>
 5  #include <sys/socket.h>
 6  #include <netinet/in.h>
 7
 8  int main()
 9  {
10      int fd, clientfd;
11      socklen_t len;
12      struct sockaddr_in r, q;
13      char c;
14
15      if ((fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
16          perror("socket");
17          return(1);
18      }
19
20      memset(&r, '\0', sizeof r);
21      r.sin_family = AF_INET;
22      r.sin_addr.s_addr = INADDR_ANY;
23      r.sin_port = htons(2000);
24
25      if (bind(fd, (struct sockaddr *)&r, sizeof r) < 0) {
26          perror("bind");
27          return(1);
28      }
29      if (listen(fd, 5)) {
30          perror("listen");
31          return(1);
32      }
33
34      while (1) {
35          len = sizeof q;
36          if ((clientfd = accept(fd, (struct sockaddr *)&q, &len)) < 0) {
37              perror("accept");
38              return(1);
39          }
40          switch (read(clientfd, &c, 1)) {
41          case -1:
42              perror("read");
43              return(1);
44          case 1:
45              if (write(clientfd, &c, 1) != 1)
46                  perror("write");
47          }
48          close(clientfd);
49      }
50  }
```


continued...

(question 10, continued)

a) Regarding the line "r.sin_port = htons(2000);" (line 23), what would happen if instead we simply wrote "r.sin_port = 2000;"?

b) On which line number will this program be blocked when no client is connected?

c) On which line number will this program be blocked when a client is connected but has not yet transmitted its byte?

d) In the switch statement beginning on line 40, under what circumstances will neither of the cases match? What will read()'s return value be, and why?

e) Change the program in place so that it mostly functions as it currently does, except that if the byte transmitted from the client is a control-B (ASCII value 2), instead of sending the control-B back it sends the string "ha!" plus a network newline.

continued...

Extra space if needed
(you must write "see page 12" in the usual answer space for the given question)

End of exam. Total marks: 100. Total pages: 12.