University of Toronto
Faculty of Arts and Science

August 2022 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.

Last name:

First name(s):

Student number:

Do not open this booklet until you are instructed to.
When instructed to begin, please check that you have all 12 pages (including this one).

Please fill in your name and student number above, now.

Turn off all mobile (cell) phones, "smart watches", and other electronic devices, and place them in your bag under your desk. If you possess an electronic device during the exam, you may be charged with an academic offence.

When you are finished with the exam, raise your hand for someone to collect it. Do not access unauthorized aids before your exam is collected.

Answer *all* questions. Answer questions in the space provided, or write "see page ..." and identify the answer on that other page clearly.
You can omit comments unless you need to explain something unclear about the functioning of your code, and in the C programming questions you can omit the #includes.

Do not write anything in the following table:

question	value	grade	question	value	grade
1	12		6	8	
2	7		7	20	
3	5		8	20	
4	9		9	9	
5	10				
subtotal			total	100	

1. [12 marks]
Write shell commands to output the following. Put some thought into making your answer simple; answers will be graded on quality, not just on whether they "work".

Please be careful to write single quotes and backquotes correctly to avoid misinterpretation.

a) the product of 123 times 234

b) the larger of the files "bloof" in the current directory and "bloof" in the root directory. Output the file path name, e.g. simply "bloof" if the file in the current directory is bigger. (In the event of a tie, outputting either one file name is fine.)

c) whether the value of the variable f, plus 9, is equal to the value of the variable g (output either "yes" or "no")

d) the number of '<' characters in the standard input. (Thus, with certain simplifying assumptions, if the input is HTML, then this counts the number of HTML tags.)

2. [7 marks]
There are files named a1, a2, a3, and so on through a20.

Write a loop in *sh* to rename a1 to a2, a2 to a3, a3 to a4, and so on, through renaming a20 to a21.

3. [5 marks]
Consider the following terminal session, where '$' is the shell prompt:

```
$ cat s
echo Blah, blah, blah[footnote].
echo Blah blah, and blah blah.
echo
echo [footnote]Additionally, blah blah.
$ sh s
Blah, blah, blah[footnote].
Blah blah, and blah blah.

[footnote]Additionally, blah blah.
$
```

In this case, the shell script worked as expected; but in fact, it has a serious bug, relating to special characters. The output will not always be as above.

a) What is the bug?

b) What would trigger the bug? (That is, suppose you wanted to show a sceptical person that the bug really exists — what would you do to demonstrate its existence?)

4. [9 marks]
Recall the difference between "`tr x y`" and "`sed s/x/y/`" — this *tr* command replaces all 'x' characters but this *sed* command replaces only the first 'x' on each line.

Write a complete C program (except that you can omit the #includes) which takes no command-line arguments (you don't have to declare the argc and argv arguments to main()) and does the same transformation as "`sed s/x/y/`" — that is, stdin is copied to stdout, with the first 'x' on each line (if any) changed to 'y', but otherwise unmodified.

5. [10 marks]
Write a function in C (not a complete program) which takes an array of integers (passed as a pointer to the zeroth element plus a separate size parameter) which are file descriptors, and an array of char (again a base pointer plus a separate size parameter) which is an area to read() into.

So the arguments are much like those to read(), except we have an array of file descriptors instead of a single file descriptor.

Your function calls select() to find one of the file descriptors which is first ready for reading, and reads from that file descriptor. Other than the select() call, your function behaves similarly to read(): it reads into the char array, and returns the number of bytes read. Also like read(), in the event of error, your function does not produce an error message, but simply returns −1 (which is what read() will return to you).

Reminder: `extern int read(int fd, char *buf, int bufsize);`

```
int read_any(int *fdlist, int fdlistsize, char *buf, int bufsize)
{
```

 continued...

6. [8 marks]
Given these declarations:

```
int n;
int *p;
int x[4];
```

and given values as follows:

`n` is 5
`p` is a pointer to `x[1]`
`x[0]` is 0; `x[1]` is 1; `x[2]` is 2; `x[3]` is 3

specify the type and value of each of the following expressions. (For pointer values you can write things like "pointer to x[1]".) If the expression is invalid, write "invalid".

	type	value
`p+2`		
`p+n`		
`*p`		
`*(p+2)`		
`&p`		
`p[1]`		
`&p[1]`		
`'a'+n`		

7. [20 marks]
There is an existing function called "sthg" (short for "do something") which takes one integer argument and returns an integer. It does some sort of computation which we want to perform in parallel for various arguments.

Write a function in C (not a complete program) which takes an array of integers (as the usual two parameters: base address of the array and size of the array). It forks once per array element and, in parallel, calls sthg(i) for each array element. The results of all of the computations are added together, and this is the return value of your function.

To convey the results of the computations back to the original process, you will need to use pipes. Each child process does the computation and then writes the result to the pipe.

```
int parallel_something(int *a, int size)
{
```

(more space for your question 7 answer, if required)

8. [20 marks]
The file /dev/urandom is a special device file which is a source of random bytes, as we used early in the course in the "how much is that doggie in the window?" example, and again in lab 12. (You don't need to remember those examples to do this question.)

The file /usr/share/dict/words is a list of English words for use in spell-checking, with one word per line. It is more than 65536 lines long. Each line is fewer than 500 characters wide.

Write a complete C program (except that you can omit the #includes) which reads two bytes from /dev/urandom to get a 16-bit number from 0 to 65535 ($2^{16}-1$), and then reads /usr/share/dict/words to get a random line from the first 65536 lines of the file (from line 1 to 65536). Output that line.

(Note: You still need to check for all errors. For example, if /usr/share/dict/words can't be opened, you would need to output a correct error message. If it is fewer lines long than the random number you generate, you can simply exit without output, but your program mustn't misbehave or violate the C language rules. If a line is longer than 500 characters, you can end up splitting it in two, or losing some of it, or anything like that, but you can't exceed array bounds.)

(more space for your question 8 answer, if required)

9. [9 marks]
The following program is meant to be a server which listens on port 2000, and for each client which connects, it reads one character (byte) from the client, sends it back to that client, and drops the connection.

This program doesn't work.
(I suggest reading through the code first, rather than skipping ahead to the questions.)

```
 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;
11      socklen_t len;
12      struct sockaddr_in r, q;
13      int 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 = 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 ((fd = accept(fd, (struct sockaddr *)&q, &len)) >= 0) {
37              switch (read(fd, &c, 1)) {
38              case -1:
39                  perror("read");
40                  return(1);
41              case 1:
42                  write(fd, &c, sizeof c);
43              }
44          }
45      }
46  }
```

Questions about this code appear on the following page.

continued...

(question 9, continued)

The following bugs are observed. For each bug, state the fix, by writing the actual replacement code and saying what should be replaced or where the new code should be inserted, using the line numbers on the original code for reference (e.g. "insert this after line 12").

a) It listens on the wrong port number. For some reason it is listening on port 53255 (determined by running a tool which shows what ports have programs listening on them), rather than on port 2000.

b) It indeed echoes back the first character sent, but then also sends some seemingly-random garbage.

c) It doesn't drop the connection after doing that. (The introductory description said that this program is supposed to drop the connection after sending the character back to the client.)

d) It only works for the first client. Subsequent connections are not apparently answered. In starting to debug this, you notice that you are not handling errors from accept() (line 36 does check accept's return value, but does not do anything in the negative case). You add an 'else' after line 44 to call perror() and exit. For the second client, this outputs the error ''accept: Invalid argument''. The man page says that this means that ''socket is not listening for connections''.

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.