University of Toronto
Faculty of Arts and Science

December 2013 Examinations

CSC 209H1F

Duration: 3 hours

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

If you are writing this exam at the normal time and place, you have received a label with your seat number and name on it. Attach that label here, over this text. Please remove all of the backing.

Otherwise, please write your name and student number, underlining your surname:

Make sure you have all 9 pages (including this page).

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	12		6	11	
2	6		7	15	
3	5		8	15	
4	9		9	15	
5	12		total	100	

1. [12 marks]
Write shell commands to perform the following actions.
Please be careful to write single quotes and backquotes correctly to avoid misinterpretation.

a) Output the contents of a file named: `@$%*`

b) Use *who* and other tools to find out who logged in at 10:45 (there is only one), but output only the login name, not the entire 'who' line. (The login name is first on the line, and is followed by at least one space.)

c) The variables 'x' and 'y' contain integer values (as strings, of course; everything in *sh* is a string). Set the variable 'z' to either the value of 'x' or the value of 'y', whichever is greater.

d) The questions on an exam are stored in files named q1, q2, q3, and so on up to q10. I decide to insert a new question as question 6. Write a 'for' loop to rename the files from q6 through q10 to the new names q7 through q11 respectively (so that I can then create the new q6).

e) A natural number is simply a string of one or more digits. You are cd'd to a directory containing files which contain a mixture of natural numbers and other data (e.g. there might be a line which says "I deleted the file 'hello123' at 12:45" — that contains the numbers 123, 12, and 45). But we're not interested in all of the files in this directory. Output the greatest natural number out of the contents of all files whose names end with an odd digit (e.g. including a file named foo321, but not foo312 nor simply foo).

 continued...

2. [6 marks]
Someone writes an *sh* shell script which contains the line:

```
echo ‘expr $i + 5‘
```

(The variable ‘i’ is known at this point in the program to contain a valid integer value.)

a) What does this command do?

b) Write a simpler equivalent command.

3. [5 marks]
Here is a C program:

```
int main()
{
    return(3);
}
```

It is compiled and the result is put into the file `/u/betty/x`
Then this shell script is executed:

```
if /u/betty/x
then
    echo hello
else
    echo goodbye
fi
```

a) What is the output of the shell script?

b) Change the C program in place above so that the shell script would produce the other output.

4. [9 marks]
Euclid's greatest-common-divisor algorithm can be expressed in a C-like language as:

```
int gcd(int x, int y)
{
    int t;
    while (y > 0) {
        t = x;
        x = y;
        y = t % y;
    }
    return(x);
}
```

Write a complete shell script which computes the gcd of all of the numbers on the standard input (and outputs the gcd value to the standard output). You can assume that the values on standard input are all non-negative integers, one per line, and that there is nothing else in the file.

Similar to summing a list of numbers, you can start with zero and repeatedly use Euclid's algorithm to compute the gcd of the new number with the cumulative gcd of all of the numbers so far.

Your shell script will ignore its command-line arguments.

Note that the *expr* command has a '%' operator.

5. [12 marks]
Write a complete C program (except that you can omit the #includes) which attempts to fopen() files named "1", "2", and so on up to "100" inclusive. When the fopen() succeeds, it checks whether the first character in the file is a '#'. When the fopen() fails, it just loops around to the next file without emitting any error message (i.e. it's ok for these files not to exist). At the end of your program's execution, it outputs the total count of files (out of those examined) which do indeed have an '#' as their first character.
(You can continue your answer onto the next page.)

 continued...

6. [11 marks]
In C, write a simple version of the unix *cp* command which takes no options, just the two plain file names. (And in your simplified cp, the user can't say "cp file dir" to create dir/file. The second argument is always the target itself.)

7. [15 marks]
The FILE type declaration in stdio.h looks something like this (although with more fields):

```
struct file {
    char buf[BUFSIZ];
    int bytes_in_buf;  /* amount of unwritten data currently in buf */
    int fd;  /* the unix file descriptor for this open file */
    int perms;  /* whether open for read or write -- 0 for read, 1 for write */
    int error;
};
typedef struct file FILE;
```

a) Write the putc() library function. This function adds the character to the 'buf' buffer. If the 'buf' buffer is already full, it first calls write() to output the full buffer to the file.

putc() returns an int which is its 'c' argument if it succeeds, or –1 if it fails. (The only way it can fail is if write() fails.) If the write() fails, putc() also sets the ''error'' flag for later use by ferror() and fclose(). (fopen() has initialized this to zero.)

(putc() assumes that the file pointer is open for write; you don't need to check 'perms'.)

```
int putc(int c, struct file *fp)
{
```

b) Write the fclose() library function. This has to deal with pretty much all of the elements of the struct above, except that data in the buffer in the case of a file open for read (see the 'perms' value) can simply be discarded (ignored). And the FILE* argument itself was originally a pointer returned by malloc() inside fopen(); this also needs to be cleaned up appropriately. This function returns an int, which is –1 if an error occurred at any point since the fopen() (including during fclose()), or 0 otherwise.

```
int fclose(struct file *fp)
{
```

 continued...

8. [15 marks]
Write a C program which executes every file in the current directory (except for '.' and '..'). Your program should fork and execute each file so that they run in parallel. Once all of these processes have been spawned, your program can exit (you don't have to wait() for them). (You should not attempt to check in advance whether the file is executable; just run it, and display appropriate error messages if the system call indicates an error condition.)
You can omit the #includes.

9. [15 marks]
Write a C program which listens on port number 2345 and repeatedly accepts connections. For each connection, it reads all of the data sent by the other side (until the other side drops the connection, i.e. you get end-of-file) and outputs it to stdout. (Once the other side drops the connection, your program loops around and accepts a new client.)

You do not have to handle multiple simultaneous incoming connections.
You do not have to worry about the difference between the host and network newline conventions.
You can omit the #includes, and you can cite course web pages rather than copying out bits of code if you can do so unambiguously.

Extra space if needed
(you must write "see page 9" in the usual answer space for the given question)

End of exam. Total marks: 100. Total pages: 9.