UNIVERSITY OF TORONTO
Faculty of Arts and Science

AUGUST 2006 EXAMINATIONS

CSC 209 H1 Y

Duration — 3 hours

Examination Aids: One letter sized (8.5" × 11") sheet of paper
No electronic aids allowed: no cell phones, calculators, computers, etc.

Student Number: |_|_|_|_|_|_|_|_|_|

Last (Family) Name(s): ______________________________

First (Given) Name(s): ______________________________

---

*Do **not** turn this page until you have received the signal to start.*
(In the meantime, please fill out the identification section above, and read the instructions below *carefully*.)

This term test consists of 8 questions on 13 pages (including this one). The last page includes C function prototypes and `sh` shell usage notes that might be useful: you may detach and keep this sheet. *When you receive the signal to start, please make sure that your copy of the test is complete.*

Answer each question directly on the test paper, in the space provided. If you need more space for one of your solutions, *indicate clearly the part of your work that should be marked and which question it answers.*

You do not need to include the "`#!`" line in Bourne shell scripts you are asked to write. In C programs, you do not need to add the "`#include`" lines, nor do error checking unless the question requires it or the program would not function correctly given valid input without error checking.

Though you should attempt to write programs using correct syntax, *minor* syntactical errors will not be penalized.

MARKING GUIDE

# 1:	______/10
# 2:	______/16
# 3:	______/ 8
# 4:	______/ 8
# 5:	______/ 6
# 6:	______/10
# 7:	______/ 8
# 8:	______/22
TOTAL:	______/88

Good Luck!

Question 1. True or false. [10 MARKS]

Part (a) [5 MARKS]

Circle the correct answer for the questions below (1 mark each).

TRUE FALSE A *shell* is a user's interface to the Unix system.

TRUE FALSE The Unix operating system (kernel) manages access to the computer's hardware.

TRUE FALSE *System function calls* are a C program's interface to the Unix kernel and the computer's hardware.

TRUE FALSE Finding the length of a string in C is a good example of a *system function call*.

TRUE FALSE The term "Standard C Library" commonly refers to a set of 5 essential reference books that explain how to use the C programming language.

Part (b) [5 MARKS]

Circle the correct answer for the questions below (1 mark each).

TRUE FALSE In C, with the declaration `int a[10]`, the type of `a` is `int *`.

TRUE FALSE The `fork()` function (or a function that calls `fork()`) is the only way a new process can be created by a C program.

TRUE FALSE The `exec` family of functions allow us to run another program temporarily and return back to running our original program.

TRUE FALSE Unrelated processes can communicate with each other using signals.

TRUE FALSE Sockets allow programs on different machines to communicate with each other.

Question 2. Short answer. [16 MARKS]

Part (a) [2 MARKS]

Write a Bourne shell (`sh`) command that outputs the product of 209 and 369. Use as few processes as possible.

Part (b) [2 MARKS]

Write a Bourne shell (`sh`) command that outputs the number of different settings of default shells being used on CDF. HINT: the default shell is the 7th "`:`"-delimited field in `/etc/passwd`. The programs `sort` and `uniq` may also be useful.

Part (c) [2 MARKS]

Write C statement(s) to allocate space for an array of 5 signed integers such that the space is automatically deallocated when it goes out of scope.

Part (d) [2 MARKS]

Write C statement(s) to allocate space for an array of n signed integers, where n is a variable of type `int`.

Part (e) [2 MARKS]

Write an equivalent expression in C for the expression "`*(a + 3) = 10;`" where `a` is declared as `int a[5];`.

Question 2. (CONTINUED)

Part (f) [3 MARKS]

Suppose I have two processes communicating over a pair of pipes (one pipe for each direction). Process A sends a character to process B using `fprintf(outfd, "%c", c)` which process B reads with `fscanf(infd, "%c", &c)`. Process B should then reply with a character sent to process A using similar code. The pipes and file descriptors are correctly set up.

The problem is that they both stop running in the middle of this code and never proceed to or complete the "reply" phase. What could be an explanation for this problem?

Describe briefly in English one way to fix this problem (you do not have to write the code).

Part (g) [3 MARKS]

Write the body of a C function that takes a string `s` and a character `c` as arguments and returns true if `c` appears in the string `s`, and returns false otherwise. Do not use any other functions (no library functions).

```
int strhas(const char *s, char c) {
```

Question 3. [8 MARKS]

Some programs change their behaviour based on the program name with which they were called. For example, if the `grep` program is executed using the program name `egrep`, it allows extended regular expressions instead of the traditional regular expressions (`egrep` may be a hard or soft link to `grep`). Another example is `bash`, to which `sh` is typically linked. If `bash` is executed using the name `sh` it operates in a stripped-down mode (like the real old `sh` program).

Write a Bourne shell (`sh`) script called `chameleon` that changes its behaviour based on how it's called. If your script is invoked with the name `foo` (*i.e.*, the user used the command line "`foo`" which ran your script), you should call the shell function `do_foo`. If your script is invoked with the name `bar`, you should call the shell function `do_bar`. Assume these functions exist (do not write them!). Otherwise (when invoked with any other name), attempt to run the program of the same name in either the `/bin` or `/usr/bin` directory (you may assume `chameleon` is not located in these directories). In all cases, pass *all* command line arguments on to the new program or shell function.

HINTS: Set your path appropriately. Recall that `basename` strips the directory from filenames.

Question 4. [8 marks]

Write a Bourne shell (`sh`) script called `termall` that attempts to terminate all processes with process ID between 1 and 10000. (Of course, it will only succeed for processes owned by the current user.) The `SIGTERM` signal should be used.

Hint: Don't kill yourself too early!

Question 5. [6 marks]

Write the body of the `strunixtonet()` C function below. This function should write the contents of the string `s` to the socket `fd`, translating the string from Unix line-ending format to network line-ending format, *i.e.*, converting `'\n'` characters to the `'\r\n'` combination. This function does not modify `s` and returns the number of bytes written to `fd`.

```
int strunixtonet(int fd, const char *s) {
```

Question 6. [10 MARKS]

The “message of the day” (motd) is printed to a user after a successful login. CDF uses this to display lab information and remind users not to eat or drink in the labs. It is typically also used to display information about outages or changes to the systems.

Write a C program that will display the `/etc/motd` file without having a user log in. When a client connects to TCP port 17 (the “Quote of the Day” port), your program should send the contents of `/etc/motd` to the client (in network line-ending format as described in Question 5), then disconnect.

You may assume that the `/etc/motd` exists and is readable by all users. You may use the `strunixtonet` function from the previous question in your solution.

Question 7. [8 marks]

Suppose we have a program `x` that only takes input from standard input. If we want the input to come from a file instead, we could easily use shell I/O redirection. The problem comes when we want `x` to run on several files: we would need to write a loop to run `x` with input redirected from each file.

Write a Bourne shell (`sh`) script called `wrap` that takes a program, any switches to that program (each starting with a `-`), and a list of files (none start with a `-`) to iteratively run on. For example,

```
wrap x -z a b c
```

would run `x -z` three times, with input reading from `a`, `b` and `c`.

HINT: The command "`expr -z : -`" prints `1`.

(this page is for any answers that didn't fit in the space provided)

Question 8. [22 MARKS]

You will write a program in C named `cat2` that takes two command lines as arguments, executes both commands in parallel, and combines the output of each to standard output (in a particular way, described now). Whenever a byte of output is available from one program but not the other, it is printed to stdout. Whenever bytes are available from both programs, a byte from the first program is printed followed by a byte from the second program.

For example, if program `a` prints 5 a's and program `b` prints 5 b's, `cat2 "a" "b"` might output `aaabababbb`.

You may assume you are given two command lines as `argv[1]` and `argv[2]`, but you may not assume that a command line can be executed successfully. If one command line cannot be executed, the behaviour of `cat2` should be identical to the `cat` command.

Part (a) [4 MARKS]

Write a high-level overview of your algorithm. Mention which system calls you use in each step. For example, your algorithm might begin with the following line:

```
create two pipes
```

Part (b) [3 MARKS]

Assume you are given a function `char **break_into_args(char *s)` that takes a command line string and returns a pointer to an `argv` array representing this command line.

Write the C code the child process should execute to run the command line passed in as `s`. Assume all file descriptors are already set up for you. Remember that the `execvp` call is not guaranteed to succeed!

```
void do_child(char *s) {
```

Question 8. (CONTINUED)

Part (c) [7 MARKS]

Write the C code for the first part of the `main` function. You should not do any reading or writing, but should do everything up to that point. The child processes should execute the `do_child` function with an appropriate argument at the appropriate time. Make sure you close any unused file descriptors.

You may use the comment "`/* now do likewise for second child */`" to replace the second `fork` call and reduce how much you must write.

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

```
/* postconditions: Child processes have been started, all unused file descriptors
 *   have been closed. Ready to start reading and echoing! */
```

Question 8. (CONTINUED)

Part (d) [8 MARKS]

Write the C code the parent will execute to read output from the child processes and output to stdout such that whenever a byte of output is available from one program but not the other, it is printed to stdout, and whenever bytes are available from both programs, a byte from the first program is printed followed by a byte from the second program. HINT: Use `read` to read one byte at a time.

```
/* preconditions: Child processes have been started, file descriptors are set up
 *   properly. Ready to read from fd's ________ and ________ to echo to stdout. */
```

Total Marks = 88

(this page is for any rough work or answers that didn't fit in the space provided)

```
int accept(int sock, struct sockaddr *addr, int addrlen);
int bind(int sock, struct sockaddr *addr, int addrlen);
int close(int fd);
int closedir(DIR *dir);
int connect(int sock, struct sockaddr *addr, int addrlen);
int dup2(int oldfd, int newfd);
int execlp(const char *file, char *argv0, ..., (char *)0);
int execvp(const char *file, char *argv[]);
int fclose(FILE *stream);
int FD_ISSET(int fd, fd_set *fds);
void FD_SET(int fd, fd_set *fds);
void FD_CLR(int fd, fd_set *fds);
void FD_ZERO(fd_set *fds);
int fgetc(FILE *stream);
char *fgets(char *s, int n, FILE *stream);
int fileno(FILE *stream);
pid_t fork(void);
FILE *fopen(const char *file, const char *mode);
int fprintf(FILE *stream, const char *format, ...);
int getchar(void);
struct hostent *gethostbyname(const char *name);
unsigned long int htonl(unsigned long int hostlong);
unsigned short int htons(unsigned short int hostshort);
char *index(const char *s, int c);
int kill(int pid, int signo);
int listen(int sock, int n);
unsigned long int ntohl(unsigned long int netlong);
unsigned short int ntohs(unsigned short int netshort);
int open(const char *path, int oflag); /* O_RDONLY, O_WRONLY or O_RDWR */
DIR *opendir(const char *name);
int pclose(FILE *stream);
int pipe(int filedes[2]); /* filedes[0] for reading, filedes[1] for writing */
FILE *popen(char *cmdstr, char *mode);
int putchar(int c);
ssize_t read(int fd, void *buf, size_t count);
struct dirent *readdir(DIR *dir);
int select(int n, fd_set *readfds, fd_set *writefds, fd_set *exceptfds,
           struct timeval *timeout);
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
    /* actions include SIG_DFL and SIG_IGN */
int sigaddset(sigset_t *set, int signum);
int sigemptyset(sigset_t *set);
int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
    /* how has the value SIG_BLOCK, SIG_UNBLOCK, or SIG_SETMASK */
unsigned int sleep(unsigned int seconds);
int socket(int family, int type, int protocol);
    /* family is PF_UNIX, PF_INET; type is SOCK_STREAM, SOCK_DGRAM */
int sprintf(char *s, const char *format, ...);
```

```
int stat(const char *file name, struct stat *buf);
char *strchr(const char *s, int c);
size_t strlen(const char *s);
char *strncat(char *dest, const char *src, size_t n);
int strncmp(const char *s1, const char *s2, size_t n);
char *strncpy(char *dest, const char *src, size_t n);
char *strrchr(const char *s, int c);
char *strstr(const char *haystack, const char *needle);
int wait(int *status);
int waitpid(int pid, int *stat, int options); /* options = 0 or WNOHANG*/
ssize_t write(int fd, const void *buf, size_t count);

WIFEXITED(status)       WIFSIGNALED(status)     WIFSTOPPED(status)
WEXITSTATUS(status)     WTERMSIG(status)        WSTOPSIG(status)

struct hostent {
   char *h_name;     /* official name of host */
   char **h_aliases; /* alias list */
   int h_addrtype;   /* host address type */
   int h_length;     /* length of address */
   char *h_addr;     /* address */
};

struct sigaction {
   void (*sa_handler)(int);   /* function or SIG_DFL or SIG_IGN */
   sigset_t sa_mask;
   int sa_flags;     /* SA_NOCLDSTOP, SA_RESTART, SA_NOMASK, etc. */
};

struct sockaddr_in {
   sa_family_t    sin_family; /* AF_INET */
   u_int16_t      sin_port;
   struct in_addr sin_addr;
   unsigned char  sin_zero[8]; /*Unused*/
};
```

Shell variables:

`$$`	shell process ID
`$?`	last program exit status
`$#`	number of arguments
`$*`	all arguments as string
`"$@"`	all arguments as quoted list

```
cut [-d X] [-c CHARS | -f FIELDS]
expr EXPRESSION
grep pattern file ...
kill [-s sig] pid ...
for ... in ... do ... done
while ... do ... done
```

Shell test comparison operators:

Shell	Description
-d filename	Exists as a directory
-f filename	Exists as a regular file
-r filename	Exists as a readable file
-w filename	Exists as a writable file
-x filename	Exists as an executable file
-z string	True if empty string
str1 = str2	True if str1 equals str2
str1 != str2	True if str1 not equal to str2
int1 -eq int2	True if int1 equals int2
-ne, -gt, -ge, -lt, -le	Comparisons for numbers
!=, >, >=, <, <=	Comparisons for strings
-a, -o	And, or