

PLEASE HAND IN

UNIVERSITY OF TORONTO  
Faculty of Arts and Science

St. George Campus

APRIL 2015 EXAMINATIONS

CSC 209H1S

Instructors:

Karen Reid and Bogdan Simion

Duration: 3 hours

PLEASE HAND IN

Examination Aids: One double-sided 8.5x11 sheet of paper. No electronic aids.

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.)*

---

MARKING GUIDE

# 1: \_\_\_\_\_/10

# 2: \_\_\_\_\_/ 6

# 3: \_\_\_\_\_/ 6

# 4: \_\_\_\_\_/10

# 5: \_\_\_\_\_/ 6

# 6: \_\_\_\_\_/10

# 7: \_\_\_\_\_/ 7

# 8: \_\_\_\_\_/ 6

# 9: \_\_\_\_\_/ 8

# 10: \_\_\_\_\_/ 5

This final examination consists of 10 questions on 16 pages. A mark of at least 29 out of 74 on this exam is required to pass this course. *When you receive the signal to start, please make sure that your copy of the examination is complete.*

You are not required to add any `#include` lines, and unless otherwise specified, you may assume a reasonable maximum for character arrays or other structures. For shell programs, you do not need to include the `#!/bin/sh`. Error checking is not necessary unless it is required for correctness.

Answers that contain a mixture of correct and incorrect or irrelevant statements will not receive full marks.

*Good Luck!*

TOTAL: \_\_\_\_\_/74

**Question 1.** [10 MARKS]

TRUE FALSE You can send a signal to a process you do not own.

TRUE FALSE You can send a signal to a process that is not related to the process sending the signal.

TRUE FALSE The function `htonl()` is used to convert a string to a port number.

TRUE FALSE The `bind` call associates a port number with the server process.

TRUE FALSE When a process running in the background writes to standard output, the output appears on the screen.

TRUE FALSE The following statements are valid C and the pointer `q` does not need a type cast:

```
void *q;  
char *p = q;
```

TRUE FALSE The following statements are valid C and the pointer `q` does not need a type cast:

```
int *q;  
char *p = q;
```

TRUE FALSE A header file is used to define types, function prototypes, and global variables.

The following makefile applies to the last two sub-questions:

```
testing : input  
    countprocs input > testing  
input :  
    ps aux > testing.in
```

TRUE FALSE The actions in the rule `testing` will always be executed when `make testing` is run.

TRUE FALSE The actions in the rule `input` will always be executed when `make testing` is run.

**Question 2.** [6 MARKS]**Part (a)** [3 MARKS]

Complete the following C function that returns the number of bits that are set to 1 in the argument `set`

```
int count_ones(unsigned int set) {
```

**Part (b)** [3 MARKS]

Without using the `wc` program, write a bash shell program that prints the number of C source code files (files that end in ".c") in the current working directory.

**Question 3.** [6 MARKS]**Part (a)** [1 MARK]

Give one line you could type at the bash shell prompt to send the QUIT signal to the process with id 718.

**Part (b)** [1 MARK]

Give one line you could type at the bash shell prompt to retrieve the number of processes being run by the user "t3ta".

**Part (c)** [2 MARKS]

Write a conditional statement in bash that checks if a folder "Assignment1" exists. If the folder does not exist, it creates it, otherwise it prints "Assignment1 exists".

**Part (d)** [2 MARKS]

Check the following system calls that could cause a process to block.

☐ connect()

☐ open()

☐ strncpy()

☐ read()

☐ socket()

☐ select()

**Question 4.** [10 MARKS]

Each example below contains an independent code fragment. In each case there are variables *x* and *y* that are missing declaration statements. In the boxes to the right of the code write declaration statements so that the code fragment would compile and run without warnings or errors. If there is no declaration that could lead to a compilation without warning or errors, write "ERROR". The first is done for you as an example.

Code Fragment	Declaration for x	Declaration for y
<pre>x = 10; y = 'A';</pre>	<pre>int x;</pre>	<pre>char y;</pre>
<pre>double length = 25; x = &amp;length; y = &amp;x;</pre>		
<pre>char *id[6]; x = id[3]; // some hidden code id[3] = "c3new"; y = *x[3];</pre>		
<pre>char *name = "John Tory"; x = &amp;name; y = *(name+3);</pre>		
<pre>struct node {     int value;     struct node * next; }; typedef struct node List; List *head; // some hidden code x = head-&gt;next; y.value = 14; y.next = x;</pre>		
<pre>char fun(char *str, int n) {     return str[n]; }  x = fun; y = fun("hello", 1);</pre>		

**Question 5.** [6 MARKS]

In assignment 2, `freelist` was a linked list that contained the address and size of blocks that had been freed. The `freelist` is kept in increasing order by `addr`. Several students observed that adjacent blocks could be collapsed together. For example, if one block has `addr == 0x60400` and `size == 0x10` (16 in base 10), and the next block has `addr == 0x60410`, then the two blocks can be combined into one.

Given the struct below, and the above definition of a `freelist`, complete the function `coalesce` that takes a list of blocks ordered by address, and collapses all blocks that are adjacent into a single block. It returns the head of the list.

For full marks, your solution will have no memory leaks, and will free unused memory correctly.

```
struct block {
    void *addr; /* start address of memory for this block */
    int size;   /* size of the block in bytes */
    struct block *next;
};

struct block *coalesce(struct block *list) {
```

**Question 6.** [10 MARKS]

Each of the code fragments below has a problem. Explain what is wrong, and then fix the code by printing neatly on the code itself.

**Part (a)** [1 MARK]

```
if(argc > 2) {  
    char filename[32] = argv[2];  
}
```

---

**Part (b)** [2 MARKS]

```
char s[32] = "Welcome: ";  
strncat(s, argv[1], strlen(s));
```

---

**Part (c)** [1 MARK]

```
char s[5] = "bears";
```

---

**Part (d)** [2 MARKS]

```
int sum_array(int *A) {  
    int i, sum = 0;  
    for (i=0; i< sizeof(A); i++)  
        sum += A[i];  
    return sum;  
}
```

---

**Part (e)** [2 MARKS]

```
int status;  
wait(&status);  
printf("status is %d", WEXITSTATUS(status));
```

---

**Part (f)** [2 MARKS]

```
void free_list(struct node *list) {  
    while(list != NULL) {  
        free(list);  
        list = list->next;  
    }  
}
```

**Question 7.** [7 MARKS]

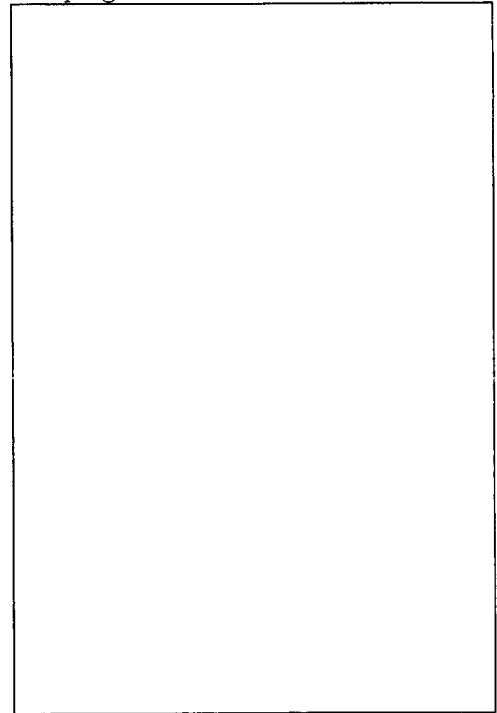
Consider the following program that runs to completion without error.

```
int main() {
    int var = 1;
    int status;

    int r = fork();
    if(r == 0) {
        var++;
        r = fork();
        if(r == 0) { //process X
            var++;
            exit(var);
        } else { // process Y
            if(wait(&status) != -1) {
                if(WIFEXITED(status)) {
                    printf("A %d ", WEXITSTATUS(status));
                }
            }
            var += 2;
        }
    } else { // process Z
        printf("W %d ", var);
        if(wait(&status) != -1) {
            if(WIFEXITED(status)) {
                printf("B %d ", WEXITSTATUS(status));
            }
        }
    }
    printf("C %d ", var);
    return 0;
}
```

**Part (a)** [3 MARKS]

Write all the possible output orders for this program.

**Part (b)** [2 MARKS]

If all processes run to completion, can process X or process Y become an orphan? If yes, explain the sequence of events that would cause the process to become an orphan. If no, then explain what modification would need be made to the code so that a process might become an orphan. Clearly identify which process is the orphan.

**Part (c)** [2 MARKS]

If all processes run to completion, can process X or process Y become a zombie? If yes, explain the sequence of events that would cause the process to become a zombie. If no, then explain what modification would need be made to the code so that a process might become a zombie. Clearly identify which process is the zombie.



**Question 8.** [6 MARKS]

In Assignment 3 students encountered the errors described in part (a) and (b). Explain the most likely reason for each error.

**Part (a)** [1 MARK]

When I enter the following command, I notice that the exec call for both processes worked correctly, but the output does not appear, and the shell does not accept any more input, but does not terminate.

```
ls -l | wc
```

**Part (b)** [1 MARK]

A simple command works fine, but when I run a command with a pipe in it, the shell prompt appears before the output of the process.

**Part (c)** [4 MARKS]

We now want to implement a new builtin command: echo. Complete the function below so that echo will print its arguments to stdout if cmd->out is NULL and to the filename cmd->out otherwise.

```
typedef struct simple_command_t {
    char *in, *out, *err;    /* Files for redirection, optional */
    char **tokens;          /* Program and its parameters */
    int builtin;            /* Builtin commands, e.g., cd */
} simple_command;

/* From execute_simple_command */
if(cmd->builtin == BUILTIN_CD) {
    //printf("Directory change requested\n");
    result = execute_cd(cmd->tokens);
}
if(cmd->builtin == BUILTIN_ECHO) {
    result = execute_echo(cmd);
}

int execute_echo(simple_command *cmd)
```

**Question 9.** [8 MARKS]

Complete the `check_answer` function below that takes an array of client structs. It waits for an answer from at least one client, reads the answer from the client (or clients if more than one produces an answer simultaneously), and then prints the name of each client whose response matches the expected "answer".

The function is called in the following context. All variables are correctly initialized.

```
/* Array of clients initialized with open socket descriptors.
 * Assume there are no invalid or empty entries in the clients array.
 * (All fd and name fields have valid values.) */
#define MAX 10
struct client {
    char *name; // name of the client
    int fd;     // open socket descriptor for the client
} clients[MAX];

/* answer is the expected response to the question and is initialized */
char answer;

/* Send a question to each client. Question is initialized. */
send_question(clients, question);

while( check_answer(clients, MAX, answer) == 0 ) {
    printf("No one has submitted a correct answer yet.\n");
}

/* check_answer blocks until at least one client responds. More than one
 * client may respond nearly simultaneously, so it is possible there
 * is a tie. check_answer reads the one-character message from each
 * responding client, and prints the name of each client that whose message
 * matches "answer"
 *
 * check_answer returns 0 if none of the clients who responded sent the
 * correct answer. It returns 1 if at least one client's response matched
 * answer.
 */

int check_answer(struct client *clients, int size, char answer) {
```

---

**Question 9.** (CONTINUED)

*More space for your solution.*

**Question 10.** [5 MARKS]

Recall from assignment 1 that a stereo wav file has the following format: The first 44 bytes are header information. The rest of the file is a sequence of pairs of `short ints`, where the first number in each pair is the left channel and the second is the right channel.

Complete the function below:

```
/* The function adjust_left takes three arguments.
 * infp - a file pointer open for reading at the beginning of a valid wav file.
 * outfp - a file pointer open for writing to a currently empty output file.
 * scale - The amount that each value in the left channel should be divided by.
 *         scale could be a fraction to make the left channel louder
 *
 * adjust_left reads the header from infp and writes it unmodified to outfp.
 * It divides each left-channel value (the first of each pair of shorts)
 * and write both channel values in the same order to outfp.
 */

void adjust_left(FILE *infp, FILE *outfp, double scale) {
```

This page can be used if you need additional space for your answers.

This page can be used if you need additional space for your answers.

Total Marks = 74

**C function prototypes and structs:**

```

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)
char *fgets(char *s, int n, FILE *stream)
int fileno(FILE *stream)
pid_t fork(void)
FILE *fopen(const char *file, const char *mode)
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
int fseek(FILE *stream, long offset, int whence);
    /* SEEK_SET, SEEK_CUR, or SEEK_END */
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
unsigned long int htonl(unsigned long int hostlong) /* 4 bytes */
unsigned short int htons(unsigned short int hostshort) /* 2 bytes */
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)
    /* oflag is O_WRONLY | O_CREAT for write and O_RDONLY for read */
int pclose(FILE *stream)
int pipe(int filedess[2])
FILE *popen(char *cmdstr, char *mode)
ssize_t read(int d, void *buf, size_t nbytes);
int select(int maxfdp1, 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 */
unsigned int sleep(unsigned int seconds)
int socket(int family, int type, int protocol) /* family=PF_INET, type=SOCK_STREAM, protocol=0 */
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)
int wait(int *status)
int waitpid(int pid, int *stat, int options) /* options = 0 or WNOHANG */
ssize_t write(int d, const void *buf, size_t nbytes);

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

```

## Useful structs

```

struct sigaction {
    void (*sa_handler)(int);
    sigset_t sa_mask;
    int sa_flags;
}

struct hostent {
    char *h_name; // 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 sockaddr_in {
    sa_family_t sin_family;
    unsigned short int sin_port;
    struct in_addr sin_addr;
    unsigned char pad[8]; /*Unused*/
}

struct stat {
    dev_t st_dev; /* ID of device containing file */
    ino_t st_ino; /* inode number */
    mode_t st_mode; /* protection */
    nlink_t st_nlink; /* number of hard links */
    uid_t st_uid; /* user ID of owner */
    gid_t st_gid; /* group ID of owner */
    dev_t st_rdev; /* device ID (if special file) */
    off_t st_size; /* total size, in bytes */
    blksize_t st_blksize; /* blocksize for file system I/O */
    blkcnt_t st_blocks; /* number of 512B blocks allocated */
    time_t st_atime; /* time of last access */
    time_t st_mtime; /* time of last modification */
    time_t st_ctime; /* time of last status change */
};

```

## Shell 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, -lt, -le	For numbers
!=, >, >=, <, <=	For strings
-a, -o	And, or.

## Useful shell commands:

cat, cut, echo, ls, read, sort, uniq, wc

ps aux - prints the list of currently running processes

grep (returns 0 if match is found, 1 if no match was found, and 2 if there was an error)

grep -v displays lines that do not match

diff (returns 0 if the files are the same, and 1 if the files differ)

\$0	Script name
\$#	Number of positional parameters
\$*	List of all positional parameters
\$?	Exit value of previously executed command