PLEASE HAND IN

PLEASE HAND IN

**UNIVERSITY OF TORONTO**
**Faculty of Arts and Science**

**St. George Campus**

**APRIL 2018 EXAMINATIONS**

**CSC 209H1S**
**Campbell, Craig**
**3 hours**

**No Examination Aids**

**You must earn at least 30 out of 76 marks (40%) on this final examination in order to pass the course. Otherwise, your final course grade will be no higher than 47%.**

**Student Number:** ⌊_____⌋ **UTORid:** ⌊_____⌋

**Family Name(s):** _____ **First 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 final examination consists of 10 questions on 22 pages. *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. Error checking is not necessary unless it is required for correctness or specifically requested.

*Good Luck!*

Marking Guide

| | |
|---|---|
| # 1: | _____/ 8 |
| # 2: | _____/ 8 |
| # 3: | _____/ 9 |
| # 4: | _____/ 6 |
| # 5: | _____/ 8 |
| # 6: | _____/ 9 |
| # 7: | _____/ 5 |
| # 8: | _____/ 9 |
| # 9: | _____/ 5 |
| # 10: | _____/ 9 |
| TOTAL: | _____/76 |

## Question 1. [8 MARKS]

Each code snippet below contains a problem. Briefly state what the problem is and then modify the code to address the problem. Make minimal changes so that the code works as intended. The code snippets in each part of this question are independent of each other.

### Part (a) [2 MARKS]

```
struct thing {
    struct thing *next;
};

struct thing *a = malloc(sizeof(struct thing));
a->next = malloc(sizeof(struct thing));
free(a);
free(a->next);
```

**Problem**

### Part (b) [2 MARKS]

```
char *return_s() {
    char *s = "hello world";
    return s;
}

int main() {
    char *r = return_s();
    free(r);
}
```

**Problem**

## Part (c) [2 MARKS]

```
int fd[2];
int res = fork();   // error-checking is not shown, but that is not the problem
pipe(fd);

if (res == 0) {
    close(fd[0]);
} else if (res > 0) {
    close(fd[1]);
}
```

**Problem**

## Part (d) [2 MARKS]

```
int *make_list() {
    int list[3] = {1, 2, 3};
    return list;
}

int main() {
    int *l = make_list();
    int i;
    for (i = 0; i < 3; i++) {
        printf("%d\n", l[i]);
    }

    return 0;
}
```

**Problem**

## Question 2. [8 MARKS]

Each code snippet below contains a problem. Briefly state what the problem is and then modify the code to address the problem. Make minimal changes so that the code works as intended. The code snippets in each part of this question are independent of each other.

### Part (a) [2 MARKS]

```
struct animal {
    char *name;
};

struct animal dog;
dog->name = "Fish";
```

**Problem**

### Part (b) [2 MARKS]

```
char *s = "hey";
s[1] = 'a';
printf("%s", s); //print "hay"
```

**Problem**

### Part (c) [2 MARKS]

```
char *word = malloc(6 * sizeof(char));
word = "hello";
```

**Problem**

### Part (d) [2 MARKS]

```
int *a;
*a = 3;
```

**Problem**

## Question 3. [9 MARKS]

Match each term below to the description that best describes it. (Functions are indicated with parentheses.) Note that some descriptions will not be used and each description should be used at most once.

| | | | | |
|---|---|---|---|---|
| ☐ `socket()` | ☐ `bind()` | ☐ `listen()` | ☐ `accept()` | ☐ `connect()` |
| ☐ endianness | ☐ `htons()` | ☐ port | ☐ `dup2()` | ☐ `select()` |
| ☐ pipe | ☐ errno | ☐ `wait()` | ☐ `sigaction()` | ☐ zombie |

A. a process that has completed execution, but is still waiting for its parent to accept its termination status

B. an active process whose parent has finished or terminated before it

C. an integer variable that is set by certain functions

D. assigns a name / address to a socket

E. changes action taken by a process on receipt of a specific signal

F. chooses a client to connect to

G. converts values from host to network byte order

H. converts values from network to host byte order

I. creates an endpoint for communication and returns a descriptor

J. describes byte order

K. establishes a queue for connections

L. examines descriptor sets to see if any are ready for reading, writing, etc.

M. gets a connection from the queue

N. gets data from a server

O. initiates a connection from the client

P. initiates a connection from the server

Q. listens for new activity on one existing client

R. makes a copy of an open file descriptor

S. monitors the activity of a signal

T. object that allows bidirectional data flow

U. object that allows unidirectional data flow

V. raises a signal

W. specifies a file descriptor to examine

X. specifies which process to interact with on a server

Y. suspends execution until a child has terminated

Z. suspends execution until all children have terminated

## Question 4. [6 MARKS]

Consider the following code.

```
/* Precondition: strlen(orig) == strlen(new) */
void change_airline(char** flights, int size, char *orig, char *new){

    for (int i = 0; i < size; i++) {
        if (strncmp(flights[i], orig, strlen(orig)) == 0) {
            strncpy(flights[i], new, strlen(orig));
        }
    }
}

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

    char first[8] = {'A', 'C', ' ', '1', '2', '3', '4', '\0'};
    char *flights[3];

    flights[0] = first;
    flights[1] = malloc(sizeof(char) * 8);
    strcpy(flights[1], "AC 5555");
    flights[2] = "WS 9876";

    change_airline(flights, 3, "AC", "UN");

    /* HERE */

    for (int i=0; i < 3; i++) {
        printf("%s\n", flights[i]);
    }

    return 0;
}
```

### Part (a) [1 MARK]

The code above runs without error. What does it print to `stdout`?

## Part (b) [5 MARKS]

Complete the memory model diagram to reflect the state of memory at the point in the program marked by the comment `HERE`.

| Section | Address | Value | Label |
|---|---|---|---|
| Read-only | 0x100 | | |
| | 0x104 | | |
| | 0x108 | | |
| | 0x10c | | |
| | 0x110 | | |
| | 0x114 | | |
| | 0x118 | | |
| | 0x11c | | |
| | ⋮ | ⋮ | |
| Heap | 0x23c | | |
| | 0x240 | | |
| | 0x244 | | |
| | 0x248 | | |
| | 0x24c | | |
| | 0x250 | | |
| | 0x254 | | |
| | 0x258 | | |
| | 0x25c | | |
| | ⋮ | ⋮ | |
| Stack | 0x454 | | |
| | 0x458 | | |
| | 0x45c | | |
| | 0x460 | | |
| | 0x464 | | |
| | 0x468 | | |
| | 0x46c | | |
| | 0x470 | | |
| | 0x474 | | |
| | 0x478 | | |
| | 0x47c | | |
| | 0x480 | | |
| | 0x484 | | |
| | 0x488 | | |
| | 0x48c | | |
| | 0x490 | | |
| | 0x494 | | |
| | 0x498 | | |
| | 0x49c | | |

## Question 5. [8 MARKS]

### Part (a) [5 MARKS]

The `sum_ith_integers` program takes an integer `i` and one or more binary file names as arguments. Each binary file contains a sequence of integers, and the program prints to `stdout` the sum of all the $i^{th}$ integers in each file.

For example, if you have files that contain binary representations for these numbers:

```
   file1                    file2                    file3
-----------              -----------              ------------
1 2 3 4 5 6              3 -2 400 384             8 21 8 0 ...
```

`sum_ith_integers 0 file1 file2 file3` would print 12. (12 is the sum of 1, 3, and 8, the integers at position 0 of `file1`, `file2`, and `file3`, respectively.) `sum_ith_integers 2 file1 file2` would print 403.

You may assume that the user calls the program with valid arguments, the each file exists, is readable and contains enough integers, and that the files were created on the same machine as the one that will run your program. You should not assume that integers are 4 bytes long.

Write C code to implement the `sum_ith_integers` program as specified above.

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

Why did we need the assumption that the files were created on the same machine as the one running the program? Give two reasons.

- Reason One

- Reason Two

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

The program prints the result, but could have been designed to return the result instead. What is the advantage of printing rather than returning the result?

## Question 6. [9 MARKS]

Consider the following struct definition:

```
struct town {
  int population;
  char name[MAXNAME+1];
};
```

### Part (a) [4 MARKS]

Write a function `grow` that will triple the population of a town represented by a `struct town` and append "-city" to the name. If this would cause the population to exceed the value stored in the pre-defined constant `MAXINT`, set the population to `MAXINT`. If there is not enough room for the new name, do not change it at all. If you encounter only one of these problems return 1, both of these problems return 2, and otherwise return 0.

## Part (b) [5 MARKS]

Write the main function for a program that will read a town name from the first command-line argument, a population from the second argument, create a struct to represent this town and then call `grow` on the struct. Do not use any dynamically-allocated memory. If the original name provided by the user is too long, just truncate it to take what will fit safely in the struct. You may assume that the population provided by the user is less than `MAXINT`.

You may assume that the user enters exactly two arguments and that the second one is an integer.

## Question 7. [5 MARKS]

Consider the following code:

```
int main(int argc, char **argv) {
    int i;
    int pid;
    for (i = 1; i < argc; i++) {
        pid = fork();

        if (pid == 0) {
            int n = 0;
            int j;
            for (j = 0; j < strlen(argv[i]); j++) {
                if (argv[i][j] == 'a') {
                    n++;
                }
            }
            exit(n);
        }
    }

    for (i = 1; i < argc; i++) {
        int status;
        wait(&status);
        if (WIFEXITED(status)) {
            printf("%d ", WEXITSTATUS(status));
        }
    }
    printf("***\n");
    return 0;
}
```

### Part (a) [1 MARK]

Briefly describe what the program does.

### Part (b) [1 MARK]

List all possible outputs generated by the program if it is compiled as `do_something` and run like so:
`./do_something orange blue aardvark`

## Part (c) [3 MARKS]

Make changes to this copy of the original program so that the same number of processes are forked, but the order of the output is guaranteed to match the order of the input. Also, change it so that the output now goes to `stderr` instead of `stdout`.

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

    int pid;

    for (i = 1; i < argc; i++) {

        pid = fork();

        if (pid == 0) {

            int n = 0;

            int j;

            for (j = 0; j < strlen(argv[i]); j++) {

                if (argv[i][j] == 'a') {

                    n++;
                }
            }

            exit(n);

        }

    }

    for (i = 1; i < argc; i++) {

        int status;

        wait(&status);

        if (WIFEXITED(status)) {

            printf("%d ", WEXITSTATUS(status));

        }

    }

    printf("***\n");

    return 0;

}
```

## Question 8. [9 MARKS]

Write a program in which the parent process forks a single child, and there is a pipe set up between the two processes that the child will use to write to the parent.

The child process should have a `SIGUSR1` signal handler that writes the integer 1 into the pipe whenever `SIGUSR1` is received.

The parent process should keep track of the number of times 1 has been read from the pipe. Whenever it reads a 1, the parent prints the updated count to standard output.

The program should continue to run indefinitely. You do not need to perform error checking for system calls, but you should close unneeded pipe ends.

### Part (a) [8 MARKS]

Write the program as described above.

More space for your answer to Part(a):

## Part (b) [1 MARK]

Assume the program has begun execution, and the child's pid is 21483. Write a shell command to send the `SIGUSR1` signal to the child process.

## Question 9. [5 MARKS]

Consider the man page of an imaginary system call `office_hours`. Type `ps_set` is a defined type, similar to type `fd_set` that you used with `select`.

```
OFFICE_HOURS(2)             BSD System Calls Manual           OFFICE_HOURS(2)

NAME
     PS_CLR, PS_ISSET, PS_SET, PS_ZERO, office_hours

SYNOPSIS

     void PS_CLR(ps, ps_set *psset);

     int PS_ISSET(ps, ps_set *psset);

     void PS_SET(ps, ps_set *psset);

     void PS_ZERO(ps_set *psset);

     int officehours(ps_set *prof, struct timeval window);

DESCRIPTION
     office_hours() examines the schedules for the professors in the
professor set prof to see which have office hours scheduled within the
given window from the current time.  office_hours() replaces the
professor set with the subset of professors who have office hours
in the given window.

RETURN VALUE
     office_hours() returns the number of professors from the ps_set
who have office hours in the window, or -1 if an error occurs.
If office_hours() returns with an error, the descriptor sets will be
unmodified and the global variable errno will be set to indicate the error.
```

Suppose that (like file descriptors), professors are represented by small integers and there are professors defined as follows:

```
#define MICHELLE 1
#define ANDREW 2
#define JEN 3
#define ALAN 4
...   (there are more)
```

You are only interested in office hours held by Jen or Michelle in the next 5 hours. Finish the program on the next page, so that it calls `office_hours` and then prints either the message `"Jen has office hours"` or the message `"Michelle has office hours"` or both messages as appropriate.

Demonstrate that you know how to properly check for errors on a system call by writing the code to give the conventional behaviour if `office_hours` fails.

```
int main() {

    // set up the second argument to office_hours
    // (this is done for you and you do not need to set any other fields)
    struct timeval window;
    window.interval = 5 * 60;

    // set up first argument to office_hours









    // call office_hours (use window as the second parameter)









    // print the appropriate message(s)
```

## Question 10. [9 MARKS]

### Part (a) [7 MARKS]

Suppose that students in your course have been submitting an executable named `myprog.c` and for each student, the path to each student's file is `/209repos/<UTORID>/myprog.c` where `<UTORID>` is replaced by the student's UTORID.

Your current directory contains the following 200 files: `test1.in` ... `test100.in` and `expected1.out` ... `expected100.out` where `expected*.out` is the output that should be printed to stdout from a correct implementation of `myprog.c` when called with redirected input from `test*.in`. The current directory contains other files, but none that end in `.out` or `.in`.

On the next page, write a shell program that takes a single UTORID as a command-line argument and does the following:

- creates a new subdirectory named the same as the UTORID from the command-line argument
- changes into that subdirectory
- copies the 100 test input files, the 100 expected output files, and the student's submission into that subdirectory
- attempts to compile the student's code into an executable named `myprog`
- if the compilation is successful:
  - awards the student 5 marks for compiling successfully
  - runs the student's submission on each of the 100 tests producing `actual*.out`.
  - awards 1 additional mark for each test where the student's actual output is identical to the expected output.
  - prints to stdout, the UTORID and the total marks the student earned
- if the student is not successful:
  - prints to stdout, the UTORID and a message that the code did not compile
- changes back to the original directory

Your program should not print anything other than what is listed here. In particular, it should not print compilation messages or any other output about individual files matching or not. Note that `gcc` returns 0 when compilation is successful and a non-zero value otherwise. Complete Part (a) on the next page.

### Part (b) [1 MARK]

Suppose your program from Part (a) is called `myscript.sh`. Give the command to set its permissions so that anyone can read it or run it but nobody can change it.

### Part (c) [1 MARK]

Suppose your script has been run on many UTORIDs and the current directory now contains many subdirectories with files. Give a single shell command that from your current directory will set the permissions on all the student output files (those called `actual*.out`) so that nobody can execute or change any of the files, but TAs (who are members of the group) can copy the files. Accounts that are not members of the group should not be able to copy the files. No directory permissions need to be changed.

Write your script for Part (a) here.

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

Total Marks = 76

**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 execl(const char *path, const char *arg0, ...  /*, (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)
int fprintf(FILE * restrict stream, const char * restrict format, ...);
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
int fseek(FILE *stream, long offset, int whence);
      /* whence has the value SEEK_SET, SEEK_CUR, or SEEK_END*/
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
pid_t getpid(void);
pid_t getppid(void);
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)
void *malloc(size_t size);
int open(const char *path, int oflag)
       /* oflag is O_WRONLY | O_CREAT for write and O_RDONLY for read */
DIR *opendir(const char *name)
int pipe(int filedes[2])
ssize_t read(int d, void *buf, size_t nbytes);
struct dirent *readdir(DIR *dir)
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 */
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=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)
long strtol(const char *restrict str, char **restrict endptr, int base);
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 Makefile variables:

| | |
|---|---|
| $@ | target |
| $^ | list of prerequisites |
| $< | first prerequisite |
| $? | return code of last program executed |

Useful shell commands:

cat, cd, chmod, cp, cut, echo, expr, ls, mkdir, read, sort, uniq, set
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
wc (-clw options return the number of characters, lines, and words respectively)
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 |