PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

St. George Campus

DECEMBER 2014 EXAMINATIONS

CSC 209H1F
Instructor — Michelle Craig
Duration — 3 hours

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*.)

This final examination consists of 9 questions on 17 pages. A mark of at least 30 out of 75 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.

Good Luck!

Marking Guide

1: _____/ 5
2: _____/10
3: _____/ 3
4: _____/ 7
5: _____/ 7
6: _____/ 6
7: _____/ 8
8: _____/10
9: _____/ 5

TOTAL: _____/61

Question 1. [5 MARKS]

For each task below, give one line that you could type at the bash shell prompt to accomplish the task.

Part (a) [1 MARK]

Change the access permissions of the file `runTest` so that the owner can read, write and execute it, group members can read it and execute it, and everyone else can only read it.

Part (b) [1 MARK]

Send a SIGUSR1 signal to the process with PID 715.

Part (c) [1 MARK]

Change into the parent directory of your current working directory.

Part (d) [1 MARK]

Run the program `mark_em_all` found in the current directory so that it reads standard input from the file `classlist`, saves the standard output in a file `results`, and leaves standard error going to the console.

Part (e) [1 MARK]

Execute the program `celebrate` from the directory `/usr/mcraig/bin` with the commandline arguments `presents` and `food`.

Question 2. [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. The first is done for you as an example.

Code Fragment	**Declaration for `x`**	**Declaration for `y`**
`x = 10;` `y = 'A';`	`int x;`	`char y;`

`int age = 99;`
`x = &age;`
`y = &x;`		

`char *result[2];`
`x = result[0];`
`// some hidden code`
`result[0] = "read only";`
`y = x[0];`		

`char *name = "John Tory";`
`x = &name;`
`y = *(name+3);`		

`struct node {`
`  int value;`
`  struct node * next;`
`};`
`typedef struct node List;`
`List *head;`
`// some hidden code`
`x = head->next;`
`y.value = 14;`
`y.next = x3;`		

`char fun(char *s, int n) {`
`  return s[n];`
`}`
 	
`x = fun;`
`y = fun("easy as pie",6);`		

Question 3. [3 MARKS]

The following program attempts to assign values to elements of a struct and then print them out. The boxed sections of code are two alternatives to the accomplish the same thing. Half of them don't work properly. For each set of boxes, cross off the box holding code that doesn't work. Below it provide an explanation for why the code doesn't work. There are no marks for correctly indicating broken code without giving a correct explanation of the problem.

```
struct song {
    int track;
    char title[25];
    char *album;
    double length;
};

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

    struct song s1;

    s1.track = 2;   s1->track = 2;



    // set the title of the song to "Knee Deep"

    s1.title = "Knee Deep";   strcpy(s1.title,"Knee Deep");




    // set the album to "You Get What you Give"

    s1.album = "You Get What You Give";   strcpy(s1.album, "You Get What You Give");

}
```

Question 4. [7 MARKS]

All the subquestions below concern this `Makefile` and the programs it builds.

Makefile

```
FLAGS = -Wall -g

all : mktrans do_trans

do_trans : do_trans.o list.o
        gcc ${FLAGS} -o $@ $^

mktrans : mktrans.o list.o
        gcc ${FLAGS} -o $@ $^

%.o : %.c
        gcc ${FLAGS} -c $<

do_trans.o : list.h
mktrans.o : list.h
list.o : list.h

clean :
        rm *.o mktrans do_trans
```

Initial contents of the current working directory:

```
Makefile     list.c     mktrans.c
do_trans.c   list.h     short_trans
```

Part (a) [1 MARK]

If you run `make mktrans`, what files will be created or modified?

Part (b) [1 MARK]

If immediately after doing part a, you call `make` with no argument, what files will be created or modified?

Part (c) [1 MARK]

If immediately after doing part b, you edit `list.h` and then call `make do_trans`, what files will be created or modified?

Part (d) [2 MARKS]

Write a shell command that will run `mktrans` in the background, with no arguments, and will redirect standard output to a file called `transactions` and standard error to a file called `err`.

Part (e) [2 MARKS]

The executable `do_trans` takes one argument which is the name of a transaction file. `short_trans` is such a file. Write a rule for the makefile so that when `make test` is run, `do_trans` will be compiled if necessary, and then run using `short_trans` as the argument.

Question 5. [7 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) [2 MARKS]

```
char s[32] = argv[2];
strncat(s, argv[1], strlen(s));
```

Part (b) [1 MARK]

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

Part (c) [2 MARKS]

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

Part (d) [2 MARKS]

```
int *status_ptr;;
if (wait(status_ptr) == -1) {
    perror("wait");
    exit(1);
}
```

Question 6. [6 MARKS]

In this question you will complete a program that analyzes the characters of its command line arguments. It uses a function `stats` that takes two parameters. The first is a string `s` and the second a character `c`. The function returns an array of two integers. The first is the number of times `c` occurs in `s` and the second is the number of spaces in s.

Part (a) [4 MARKS]

//Write the function here.

```
int main(int argc, char **argv) {
    int i, *result;
    for (i=1; i<argc; i++ ) {
        result = stats(argv[i], 'e');
        printf("arg %d: %d e's and %d spaces\n",i, result[0], result[1]);
    }
    return 0;
}
```

Part (b) [1 MARK]

This main function calls `stats` on each of the command line arguments and counts the number of `e`'s and the number of spaces. The program has a memory leak. Fix it by writing directly on this copy of the code.

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

    int i, *result;

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

        result = stats(argv[i], 'e');

        printf("arg %d: %d e's and %d spaces\n",i, result[0], result[1]);

    }

    return 0;
}
```

Part (c) [1 MARK]

Assume that `stats` is written correctly and the executable is named `analyze_args`. When the program is run from the shell as `analyze_args one two three`, it prints

```
arg 1: 1 e's and 0 spaces
arg 2: 0 e's and 0 spaces
arg 3: 2 e's and 0 spaces
```

Give one example of how the program could be called from the shell so that it reports at least one non-zero number of spaces.

Question 7. [8 MARKS]

For this question you will write a program that forks one child for each commandline argument. The child computes the length of the commandline argument and exits with that integer as the return value. The parent sums these return codes and reports the total length of all the commandline arguments together. For example if your program is called `spread_the_work` and is called as `spread_the_work divide the load` it prints `The length of all the args is 13`.

We have provided some parts of the code and you must work within this framework and complete the missing pieces. You do **not** need to write `include` statements.

```
int main(int argc, char **argv) {
    //declare any variables you need
    [                                                    ]

    // write the code to loop over the commandline arguments (remember to skip the executable name)
    for [(                                                )] {
        // call fork
        [                                                    ]
        if (                                                ) {   // case: a system call error
            // handle the error

    [                                                                ]        } else if (
            // child does work here

    [                                                                ]        }
    }


    // Finish the code to wait for the children and sum up the return values

[                                                                    ]

    printf("The length of all the args is %d\n",sum);
    return 0;
}
```

Question 8. [10 MARKS]

Tom has a binary file where he keeps sales records for his customers. For each customer he has a struct with many fields. Among other things, it includes the total sales for this customer and the time and date of the most recent sale.

In this question, you will complete a program that will update this customer file. The program takes two command line arguments. The first is the customer's position in the data file and the second is the amount of today's sale. Your program will read the appropriate record for this customer from the file (without reading all the other records) and then updates the total sales and the current date. The updated record needs to replace the old record so that this customer remains in the same position in the file. We have completed much of the code for you and you must work with our existing code and comments. For this question, please do not check the return values of your system calls for errors.

```
struct customer {
    // lots of other fields not shown
    double total;
    time_t last_update;
};

int main(int argc, char **argv) {
    if (argc != 3) {
        fprintf(stderr, "Usage: update position sale_amt");
        return 1;
    }

    // compute the position in the file where we will find this customer



    // open the file and read the struct for the correct customer









    // add to the total sales and change last_update to current time
```

```
    // write the updated record back to the file
    // NOTE: Do this without closing and reopening the file









    // close the file



    return 0;
}
```

Question 9. [5 MARKS]

Consider the program below which compiles and runs without error on CDF.

```
void sneeze(int signal) {
  printf("Achoo!\n");
}

int main() {
  //kill(getpid(), SIGINT);        STATEMENT A   NOTICE IT IS CURRENTLY COMMENTED OUT

  struct  sigaction sa;
  sa.sa_handler = sneeze;
  sa.sa_flags = 0;  sigemptyset(&sa.sa_mask);
  sigaction(SIGINT, &sa, NULL);

  printf("morning\n");
  kill(getpid(), SIGINT);

  sigset_t sigset;
  sigemptyset(&sigset);
  sigaddset(&sigset, SIGINT);

  printf("class\n");
  kill(getpid(), SIGINT);

  sigprocmask(SIG_BLOCK, &sigset, NULL);
  printf("my presentation\n");
  kill(getpid(), SIGINT);
  printf("glad that's over\n");
  sigprocmask(SIG_UNBLOCK, &sigset, NULL);

  printf("party!\n");
}
```

Part (a) [4 MARKS] What is the output from the program?

Part (b) [1 MARK]

What would happen if we uncommented STATEMENT A? (select one)

- ☐ The program would no longer compile.
- ☐ The program would run only until it reached that statement and then terminate before printing anything.
- ☐ The program would print one extra `Achoo` before the output shown in part a.
- ☐ The program would behave identically with or without that statement.

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 = 61

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 */
DIR *opendir(const char *name)
int pclose(FILE *stream)
int pipe(int filedes[2])
FILE *popen(char *cmdstr, char *mode)
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)
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
expr match STRING REGEXP
expr ARG1 + ARG2
set (Note:  with no arguments set prints the list of environment variables)
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