

UNIVERSITY OF TORONTO MISSISSAUGA
APRIL 2018 FINAL EXAMINATION

CSC209H5S

Software Tools and Systems Programming

Instructors: Dema, Petersen

Duration: 2 hours

Examination Aids: None

Student Number: _____

UtorID: _____

Family Name(s): _____

Given Name(s): _____

The University of Toronto Mississauga and you, as a student, share a commitment to academic integrity. You are reminded that you may be charged with an academic offence for possessing any unauthorized aids during the writing of an exam. Clear, sealable, plastic bags have been provided for all electronic devices with storage, including but not limited to: cell phones, SMART devices, tablets, laptops, calculators, and MP3 players. Please turn off all devices, seal them in the bag provided, and place the bag under your desk for the duration of the examination. You will not be able to touch the bag or its contents until the exam is over.

If, during an exam, any of these items are found on your person or in the area of your desk other than in the clear, sealable, plastic bag, you may be charged with an academic offence. A typical penalty for an academic offence may cause you to fail the course.

Please note, once this exam has begun, you **CANNOT** re-write it.

*Do **not** turn this page until you have received the signal to start.*

In the meantime, please read the instructions below carefully.

This final examination paper consists of 6 questions on 16 pages (including this one). *When you receive the signal to start, please make sure that your copy of the final examination is complete.*

1: _____/ 3

2: _____/ 6

Comments are not required, although they may help us mark your answers.

3: _____/12

Error checking is required throughout the test.

4: _____/ 6

You do not need to provide include statements.

5: _____/ 5

If you use any space for rough work, indicate clearly what you want marked.

6: _____/ 8

You may tear the API page off the back of this exam.

TOTAL: _____/40

Question 1. [3 MARKS]

Programming Tools

Part (a) [1 MARK]

How does `make` decide which targets need to be built? Your answer should discuss dependencies.

Part (b) [1 MARK]

In assignment 3, `gdb` became more difficult to use. Explain what feature of assignment 3 made `gdb` more difficult to use and explain what you had to do to continue using it effectively.

Part (c) [1 MARK]

You've just run your program `fcopy` and you see the dreaded message "Segmentation fault: 11". Briefly explain how you would use `gdb` to identify what has caused the error.

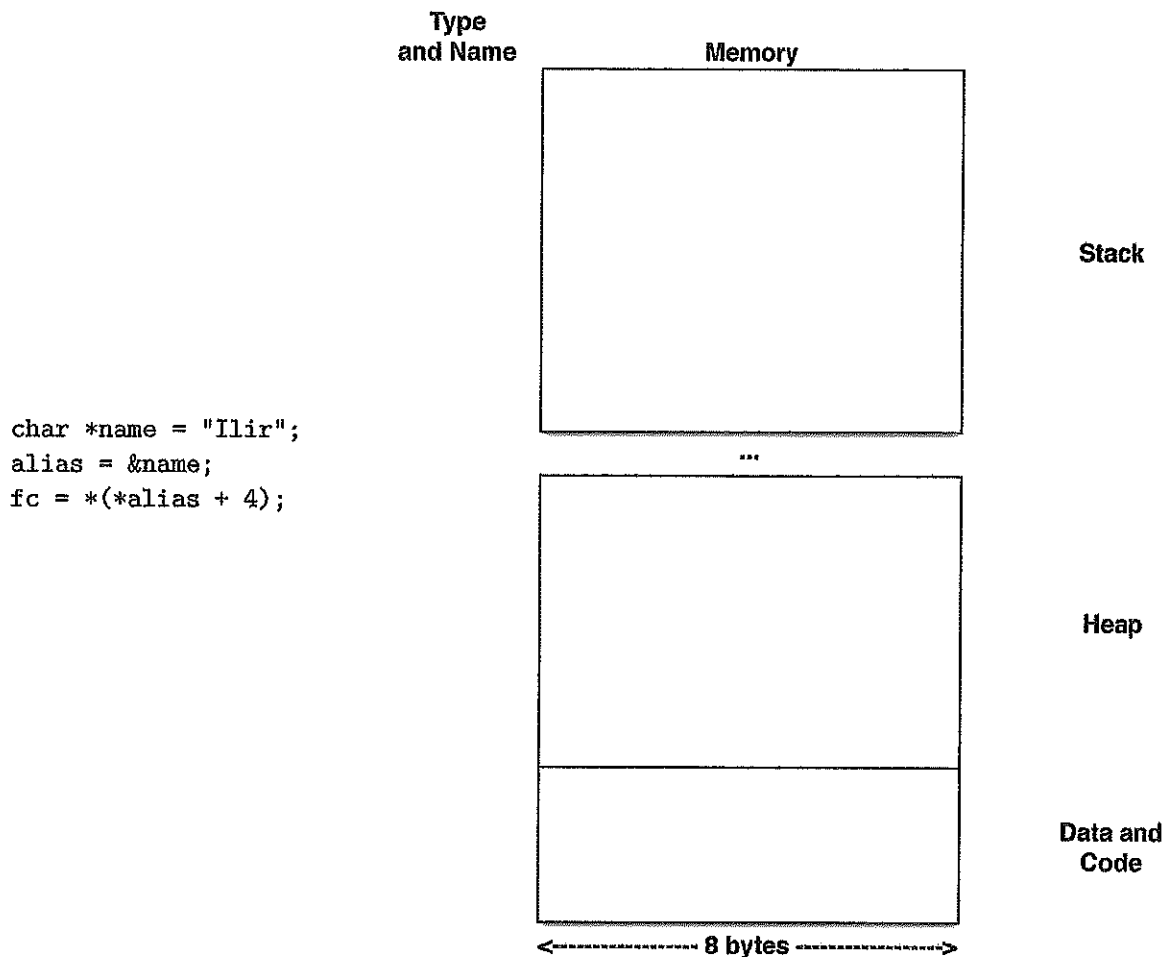
Question 2. Memory [6 MARKS]

The questions in this section feature a small piece of code and a blank diagram representing memory. Your task is to fill in the diagram so that it represents the variables and values in the code. Place the name and type of each variable to the left of the diagram, place numeric and string values into the appropriate locations in the diagram (subdividing the boxes when necessary), and represent addresses as arrows from the pointer location to the location they point to.

Pointers are 8 bytes. Integers are 4 bytes. If the value of a variable is unspecified, write "???".

In some cases, variables are mentioned without a type. You will need to infer the type of the variable but may assume that it is located on the stack.

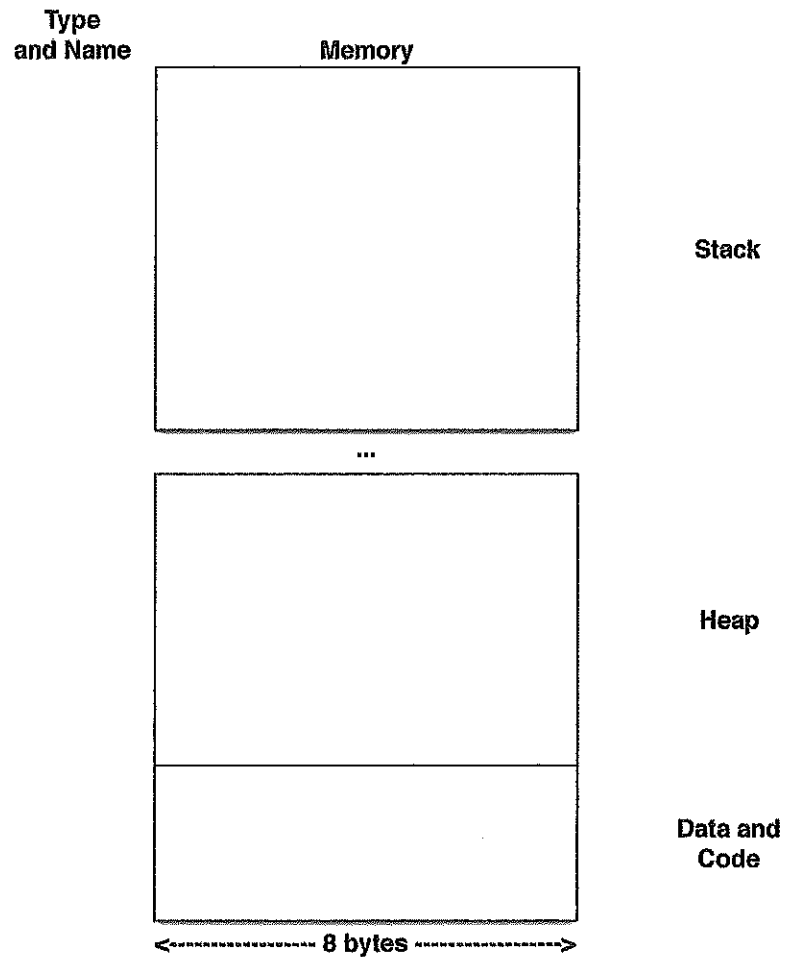
There are 8 points assigned to questions in this section, but the whole section is only worth 6. The additional points are a bonus because there are some small details that would be easy to miss.

Part (a) [4 MARKS]

Part (b) [4 MARKS]

Assume that the malloc call in the following code succeeds.

```
int vals[4] = {0, 1, 2, 3};  
i = malloc(sizeof(int) * 4);  
*i = 42;
```



Question 3. Processes [12 MARKS]

For the first two questions, consider this command which was executed in the shell:

```
./pfact 27 | grep "Number" >> num_filters.txt
```

Recall that the pipe operator redirects `STDOUT` from the process on the left to `STDIN` for the process on the right and that `">>"` redirects `STDOUT` to a file but *appends* to the file, rather than overwriting it.

Part (a) [1 MARK]

Assume that the shell process's PID is 100 and that any processes created will be assigned the PIDs 101, 102, 103, etc., and assume that `pfact` is a working instance of the assignment 3 code. How many processes are created, and for each process, what is the PID of its parent and the PID(s) of its children, if any?

Part (b) [3 MARKS]

Focus on the first `pfact` process and `grep`. Describe the work the shell process performs to *create* and *connect* the two. Name the system or library calls that are invoked, and specify their order.

Part (c) [1 MARK]

When writing A3, several people reported that the shell prompt appeared before the output of their `pfact` process. Why might that occur?

Part (d) [1 MARK]

```
int main(void) {
    int id = 0;
    for (int i = 1; i < 4; i++) {
        if (fork() == 0) {
            id = i;
        } else {
            printf("Process %d created child %d\n", id, i);
        }
    }
    return 0;
}
```

In the code above, multiple orderings of the output (`printf` statements) can be generated based on how the operating system schedules processes for execution. How many different orderings are possible? You may assume that all `fork` and `printf` calls succeed.

Part (e) [6 MARKS]

We frequently think of a file system as a tree, with each directory acting as a node in the tree. Write a program to calculate the number of subdirectories in the file tree rooted in the current working directory and to print the result. For example, if the current directory is empty or contains only normal files, the program would print "0." If the current directory contains two directories, then your program should explore both of them. If the first contains one directory (which itself contains no directories) and the second contains no subdirectories, then the program would print "3" – calculated by $(2 + (1 + 0) + (0))$.

Your program must use `fork()`. Each subdirectory should be explored using a new process. You may assume that the result will never be larger than 255 (making it safe to send results back to the parent using an exit status).

You may also assume that the helper function described above `main`, which generates an array of the names of the directories within a given directory, is available and works properly. (If you are unsure how to use this function, write a comment – `// Magic Happened` – and then assume that you have access to a magic array of strings that contains the directory names in your current working directory.)

```
// When this function returns, (*subdirs) will refer to a dynamically allocated array of
// strings. Each element in the array is the full path to a directory that is contained
// in dir. The function returns the number of strings that have been placed in the
// dynamically allocated array pointed to by subdirs.
```

```
int get_subdir_names(char *dir, char ***subdirs);
```

```
// Other helper prototypes (not required):
```

```
int main(void) {
```

Question 4. File Descriptors [6 MARKS]**Part (a) [2 MARKS]**

Consider the code below, that contains some ill-advised statements:

```
int fds[2];
pipe(fds);
if (fork() > 0) {
    fds[0] = 0;    // Yuck! Assigning 0 to the read end.
}
```

(a) Can the parent and child still communicate through the pipe, with the parent writing and child reading? Briefly justify your answer.

(b) If the child exits while the parent is writing data to the pipe, will the parent know that the child has exited? Briefly justify your answer.

Part (b) [4 MARKS]

For each of the statements below, provide one explanation of what would cause the behaviour described in the comments in each box. Please be as precise as possible.

<pre>result = read(fd, buf, SIZE); // process blocks</pre>	
<pre>result = read(fd, buf, SIZE); // result == 0</pre>	
<pre>result = read(fd, buf, SIZE); // result < SIZE but != 0</pre>	
<pre>result = write(fd, buf, SIZE); // process blocks</pre>	

Question 5. Signals [5 MARKS]**Part (a)** [1 MARK]

Some signals, like SIGINT are maskable by the users while others, like SIGKILL are not. Why?

Part (b) [4 MARKS]

Write a program that prints *"Stop poking me!"* to STDOUT and then sleeps for 5 seconds whenever the user sends a SIGINT signal. While the program is sleeping, it should not immediately respond to another SIGINT. When it is not sleeping, the program just loops (see the bottom of the page).

```
while (1);  
return 0;  
}
```

Question 6. Sockets and Select [8 MARKS]

Part (a) [1 MARK]

Why is both an address and a port required to connect to a server?

Part (b) [1 MARK]

Why does there need to be a “network order” for integers and a “network newline”?

Part (c) [4 MARKS]

Assume that an `fd_set` named `all_fds`, containing file descriptors to listen to, and an `int` named `max_fd`, containing the value of the largest file descriptor, have been defined and initialized. Write code, below, that uses `select` to print (to `STDOUT`) text that is sent on any of the descriptors. Your code *does not* need to create new connections or to remove closed connections. It also does not need to remove network newlines, but it should not print garbage.

```
// fd_set all_fds = .....;
// int max_fd = .....;
while (1) {
```

Part (d) [2 MARKS]

Assume that an `fd_set` named `all_fds`, containing file descriptors to listen to, and an `int` named `max_fd`, containing the value of the largest file descriptor, have been defined and initialized. Write code to add a new file descriptor, stored in `new_fd`, to the set, and to remove the file descriptor, stored in `old_fd`, from the set.

Total Marks = 40

C function prototypes and structs:

```

int accept(int sock, struct sockaddr *addr, int *addrlen)
char *asctime(const struct tm *timeptr)
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)
char *ctime(const time_t *clock); 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);
    /* 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);
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 filedess[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)
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);

```

Useful macros

```

WIFEXITED(status)    WEXITSTATUS(status)    FD_ZERO(fd_set)
WIFSIGNALED(status)  WTERMSIG(status)       FD_SET(fd, fd_set)
WIFSTOPPED(status)   WSTOPSIG(status)       FD_UNSET(fd, fd_set)

```

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