

UNIVERSITY OF TORONTO MISSISSAUGA
APRIL 2017 FINAL EXAMINATION

CSC209H5S

Software Tools and Systems Programming

Instructors: Chaturvedi, 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 5 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: _____ / 4

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

2: _____ / 12

No error checking is required except where indicated.

3: _____ / 12

You do not need to provide include statements.

4: _____ / 3

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

5: _____ / 9

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

TOTAL: _____ / 40

Question 1. [4 MARKS]**Programming Tools****Part (a)** [3 MARKS]

Assume that you have a project which contains the source files `rcopy_client.c`, `rcopy_server.c`, and `rcopy.c`. The project results in two executables `rcopy_client` and `rcopy_server`; each is built using the source file of the same name (with a “.c”) plus `rcopy.c`.

Write a `Makefile` that uses separate compilation to produce the two executables. (“Separate compilation” means that each source file should be compiled separately.) When `make` is run without a command line argument, both executables should be built. You may assume that there are no header files, and you do not need to write a `clean` target.

Part (b) [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. [12 MARKS]

Memory

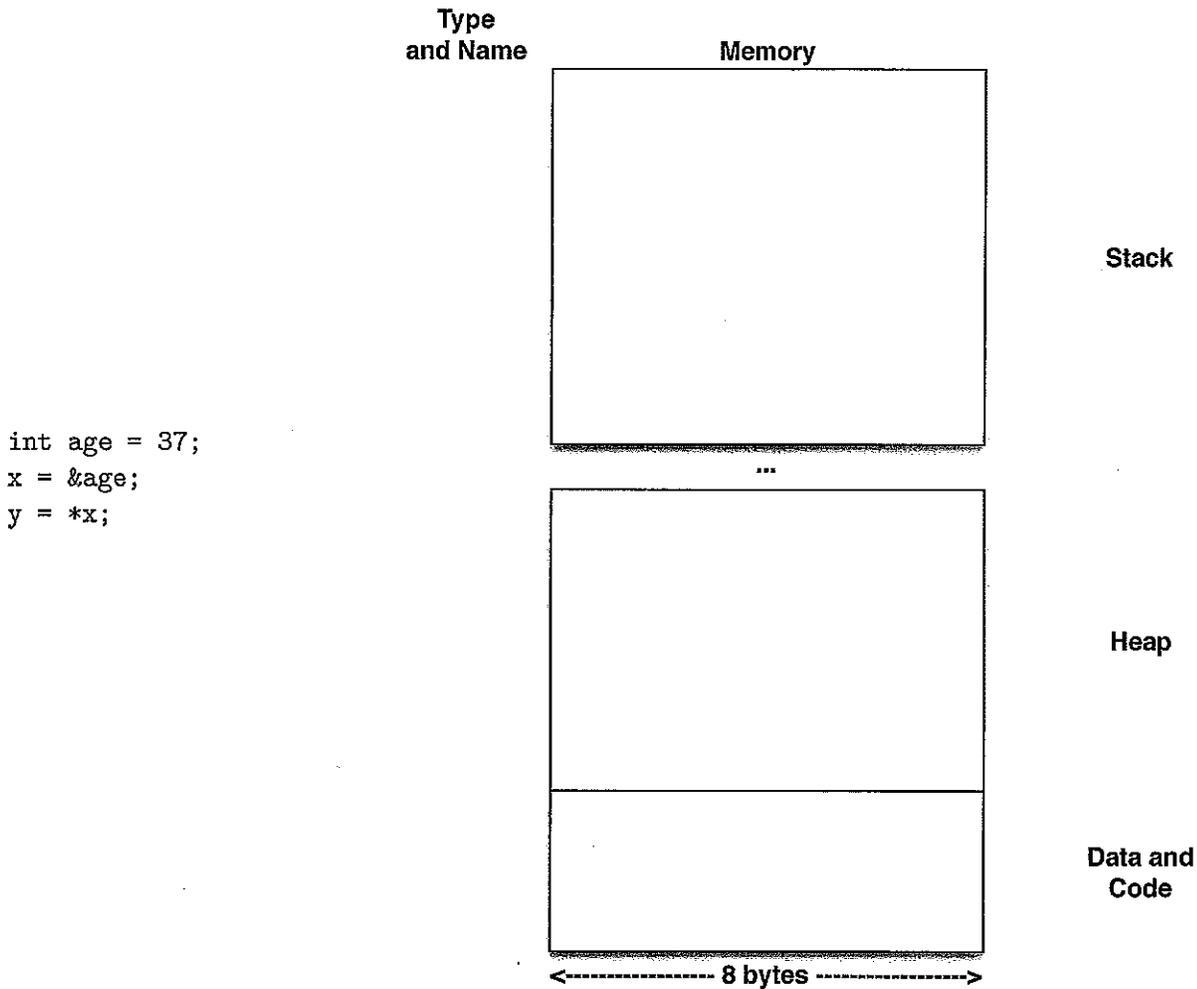
Several of the sections of this question feature a small piece of code and a blank diagram representing memory. Your task is to fill in the diagram so that it represents the variables and values in the code. Place the name and type of each variable to the right of the diagram, place numeric and string 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. 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 14 points for subquestions in this section, but the question is only worth 12. The additional 2 points are a bonus (because there are many small details that would be easy to miss).

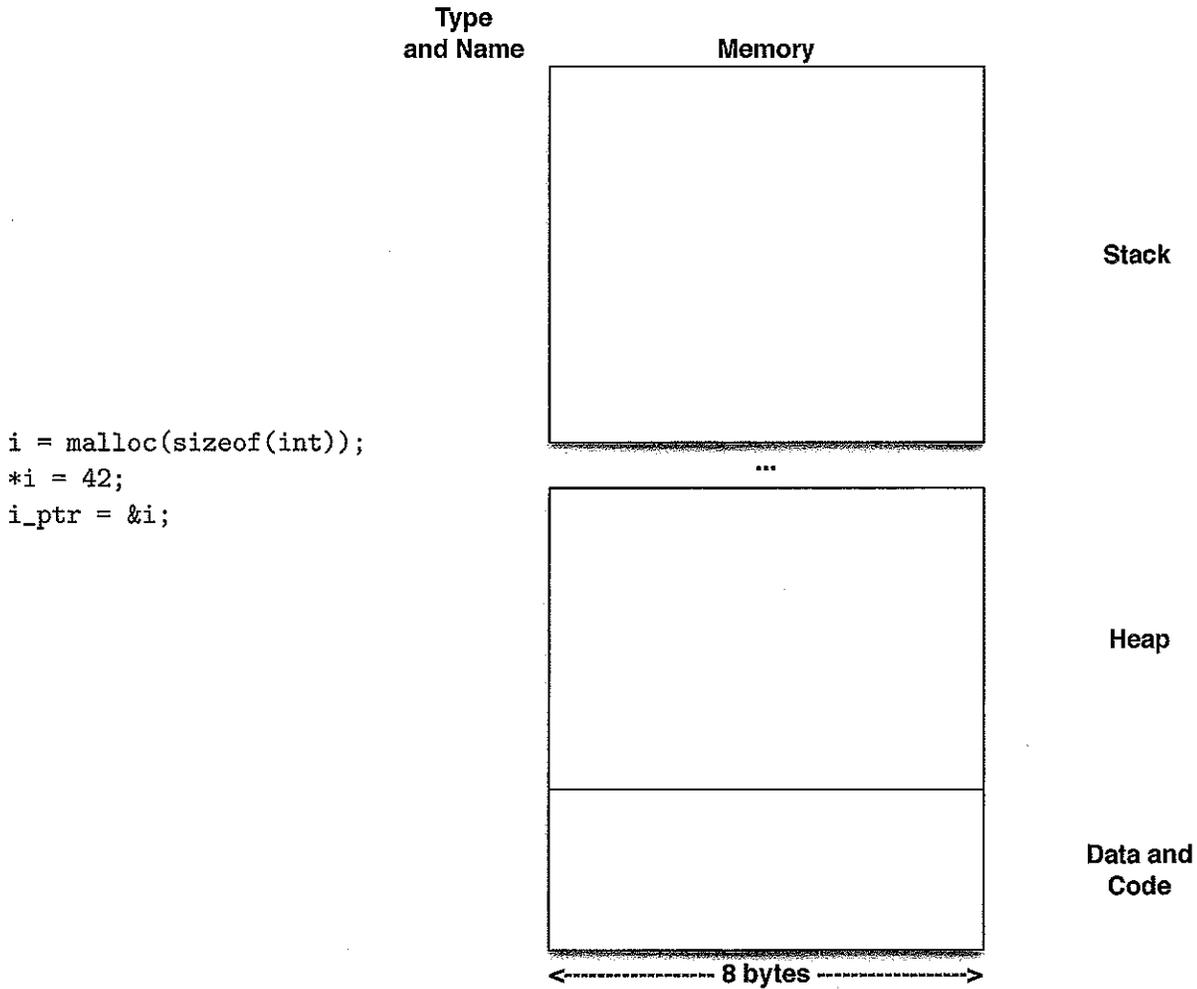
Part (a) [2 MARKS]



```
int age = 37;  
x = &age;  
y = *x;
```


Part (d) [3 MARKS]

Assume that the malloc call in the following code succeeds.



```
i = malloc(sizeof(int));
*i = 42;
i_ptr = &i;
```

Part (e) [1 MARK]

Given the following code:

```
int x;
assign(... parameters here ...);
printf("%d\n", x);
```

write a prototype for the function assign that results in the code printing the value '42'.

Question 3. [12 MARKS]**Processes, Pipe, and Exec****Part (a)** [2 MARKS]

Consider the process ID (PID), file descriptor table, and memory space to be the resources allocated to a process. Which of these resources are identical in the parent and child after the system call `fork` is called? How do the remaining resources differ between the parent and child processes?

Part (b) [2 MARKS]

Consider the process ID (PID), file descriptor table, and memory space to be the resources allocated to a process. Which of these resources remain unchanged when the system call `exec` is called? Which of these resources are modified, and how are they modified?

Part (c) [2 MARKS]

In assignment 3, we asked why *hard links* were difficult to identify. What is a hard link? Why, in a multi-process program, was it difficult to identify them?

Part (d) [1 MARK]

```
int main(void) {
    int id = 0;
    for (int i = 1; i < 3; i++) {
        if (fork() == 0) {
            id = i;
            printf("Child %d forked\n", id);
        } else {
            printf("Parent %d created child %d\n", id, i);
        }
    }
    return 0;
}
```

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

Part (e) [1 MARK]

Describe one situation where `dup2` is commonly used to duplicate the file descriptor of a pipe.

Part (f) [4 MARKS]

Write a snippet of code that (a) sets up a pipe, (b) forks, and (c) prepares the pipe so that the child can write data that the parent reads. *Please include all necessary error checks in this code and report errors appropriately.*

Question 4. [3 MARKS]

Signals

Part (a) [1 MARK]

What is a *signal*?

Part (b) [1 MARK]

Write code to install the function `timeout` as a handler for the signal `SIGALRM`.

Part (c) [1 MARK]

In the lab, why did we block `SIGALRM` while we were printing?

Question 5. [9 MARKS]

Sockets and Select

Part (a) [1 MARK]

To set up a socket for a server, you need to call `socket`, `bind`, `listen`, and `accept`. What is the purpose of the `bind` call?

Part (b) [1 MARK]

To continue from part (a), why do we need to call both `listen` and `accept`?

Part (c) [1 MARK]

Why did we need to call `htons` on the value for the `PORT`?

Part (d) [6 MARKS]

Write code for a server program. The server should accept multiple clients simultaneously and accept input from them. Whenever it receives a message from a client, it replies with the last message it received and then saves the message it just received to send to the next client who sends a message. The first client to say anything receives the message *"First"* since no previous message was received.

For example, assume two clients connect to the server. The first client sends the message *"A"*. It receives the response *"First"*. The second client closes its connection. It receives no message. A third client connects to the server. It sends the message *"B"* and receives the response *"A"*. The first client sends the message *"A again"*, and it receives the response *"B"*.

The initial server socket is set up for you, so you can focus on managing I/O. You may assume that you will never receive more than `MAX_CONNECTIONS` at one time. If two messages are received simultaneously, then the server can treat either of them as arriving first.

Please include all necessary error checks in this code and report errors appropriately.

```
#define MAX_CONNECTIONS 5
#define MAX_MSG_LEN 128

int main(void) {
    // Create the socket FD.
    int sock_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (sock_fd < 0) {
        perror("server: socket");
        exit(1);
    }

    // Code to set up the server socket (bind and listen) ...
    // You may assume this code is correct, so that sock_fd is ready to accept connections.
```

C function prototypes:

```

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 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)
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=AF_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);
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 d, const void *buf, size_t nbytes);

```

Useful macros:

```

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 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 */
};

```

Useful Makefile variables:

%	wildcard
\$@	target
\$\$	list of prerequisites
\$<	first prerequisite
\$\$?	return code of last program executed