UNIVERSITY OF TORONTO
Faculty of Arts and Science

DECEMBER 2016 EXAMINATIONS

CSC209H1F
Marina Barsky
Duration - 3 hours

Examination Aids:

1 double-sided 8.5x11 sheet of handwritten or typed notes

Last name: _____ First name: _____

Student number: _____

---

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

(Please fill out the identification section above, **write your name and student number**, and read the instructions below.)

Good Luck!

Part 1: _____/10

Part 2: _____/8

Part 3: _____/6

Part 4: _____/18

Part 5: _____/8

Part 6: _____/10

Part 7: _____/10

Total: _____/70

Part 1. Problematic code. [10 points]

All code snippets below use type *Student* defined as following:

```
typedef struct student {
     int id;
     char name [10];
} Student;
```

Would these code fragments cause a problem if used inside a larger program? Explain.
Some of the fragments may not have errors and work as intended: you do not need to give any explanation if you select the last option.
Correctly selecting one of the other options without explanations does not give any partial mark.

Code	Choice	Explanation

1.1.
`Student one;`
`one.id = 123;`
`strcpy (one.name, "Hanna");`
`if (one != NULL)`
`  printf ("id:%d, name:%s\n",`
`  one.id, one.name);`
	☐ The code **will not compile**
	☐ The code compiles but **always** gives a **run-time error**
	☐ The code causes **unpredictable behavior** and sometimes a run-time error
	☐ The code compiles, runs and **works as intended**	

1.2.
`Student two;`
`two.id = 456;`
`strcpy (two.name,`
`  "Christopher");`
`printf ("id:%d, name:%s\n",`
`  two.id, two.name);`
	☐ The code **will not compile**
	☐ The code compiles but **always** gives a **run-time error**
	☐ The code causes **unpredictable behavior** and sometimes a run-time error
	☐ The code compiles, runs and **works as intended**	

1.3.
`Student three;`
`three.id = 789;`
`char *p_name = "John";`
`three.name = p_name;`
`printf("id:%d, name:%s\n",`
`three.id, three.name);`
	☐ The code **will not compile**
	☐ The code compiles but **always** gives a **run-time error**
	☐ The code causes **unpredictable behavior** and sometimes a run-time error
	☐ The code compiles, runs and **works as intended**	

1.4.
`Student four;`
`four.id = 101;`
`four.name = "Peter";`
 	
`printf("Current name is %s\n",`
`four.name);`
	☐ The code **will not compile**
	☐ The code compiles but **always** gives a **run-time error**
	☐ The code causes **unpredictable behavior** and sometimes a run-time error
	☐ The code compiles, runs and **works as intended**	

1.5.
`Student * five_p;`
`*five_p.id = 202;`
	☐ The code **will not compile**
	☐ The code compiles but **always** gives a **run-time error**
	☐ The code causes **unpredictable behavior** and sometimes gives a run-time error
	☐ The code compiles, runs and **works as intended**	

Part 2. Shell scripts. [8 points]

Match shell commands to the output that you would see on the **screen**. Some outputs can be matched more than once, and some not at all.

Current directory contains 2 files *A* and *B*:

A	B
9 789 901	9 89

A B
AB
901
89
9 789 901 9 89
9
901 cat: D: No such file or directory
789

2.1 We execute the following command:

```
sort A B | head -1
```

2.2 We run the following bash code:

```
for i in *
do
     echo $i
done
```

2.3 Now, current directory contains a single file *C* with content "901".
What would be the output on the **screen** for the following command?

```
cat C D 2> log.txt
```

2.4 Now, current directory contains a single file *C*, the directory *D* and the following script in file *my_script*:

```
if test -d "$1"
    then
        echo 901
    else
        echo 89
fi
```

What is printed when we run:

```
./my_script C
```

Part 3. Memory problems [6 points]

Each program fragment below has problems with the memory handling. The code compiles, but often causes a run-time error. Spot the problem, explain why it is a problem, and suggest how would you fix it.

3.1. The function *free_person_list* frees the dynamic memory allocated for the linked list of Persons.

```
typedef struct person {
      char * name;
      struct person * next;
} Person;

void free_person_list (Person *head) {
      Person *temp;
      Person *node = head;
      while (node != NULL) {
            temp = node;
            node = node->next;
            free (temp);
            free (temp->name);
      }
}
```

3.2. This program fragment forks a single child process and waits until it exits. Then it prints the exit status of the child.

```
int main(void){
      pid_t pid;
      int *status;

      switch(pid = fork()) {
      case -1:
            perror("fork");
            exit(1);

      case 0:
            exit(5);

      default:
            wait(status);
            printf("PARENT: My child's exit status is: %d\n",
                                          WEXITSTATUS(*status));
      }

      … //main continues
      return 0;
}
```

3.3. The *string_copy* function produces a new copy of a given *string* and returns this copy. The problem is in the *main*.

```
char* string_copy (const char* string) {
      char* newString;
      int len;
      len = strlen(string) + 1;
      newString = malloc(sizeof(char)*len);
      strcpy(newString, string);
      return (newString);
}

int main () {
      for (int i=0; i<100000; i++) {
            string_copy ("one");
            string_copy ("two");
      }

      … //main continues
}
```

Part 4. Finish the code [18 points]

4.1. [4 points]. The following function takes a linked list of *Nodes* as input argument. It modifies the list by detaching the last element and moving it to the front of the list. It then returns the modified list. Some part of the code is left blank. Finish the function.

```
typedef struct node {
  int value;
  struct node *next;
} Node;


Node *move_to_front(Node *head) {
  Node *p, *q;
  if ((head == NULL) || (head->next == NULL))
    return head;
  q = NULL; p = head;
  while (p->next != NULL) {
    q = p;
    p = p->next;
  }
```

```
  return head;
}
```

4.2. [3 points]. The following program sorts an array of *Scores* in **descending** order of **grades**. The code is complete, only the comparator function is missing. Write the comparator to pass to *qsort* function.

```
typedef struct score {
      int student_id;
      float  grade;
} Score;
int compare_grades_desc (void *a, void *b) {

}

int main () {
    Score scores [] = {{3,90.5}, {4,40.3}, {1,84.6}};
    qsort (scores, 3, sizeof(Score), compare_grades_desc);

    puts("These are scores in descending order:");
    for (int i = 0; i < 3; i++) {
          printf("student id:%d grade:%f\n",
                      scores[i].student_id, scores [i].grade);
    }
    return 0;
}
```

4.3. [4 points]. This program implements *"ls | wc -l"* in C. Error-checking is omitted for brevity. Fill-in missing file descriptors.

```
int main(void) {
    int pfds[2];

    pipe(pfds);

    if (!fork()) {
        dup2 (________,________);
        close (________);
        execlp ("ls", "ls", NULL);
    } else {
        dup2(_________, _________);
        close (________);
        execlp ("wc", "wc", "-l", NULL);
    }

    return 0;
}
```

4.4. [4 points]. This is a code for an internet socket server, which gets a single message from a single client, and exits. Insert code snippets from page 17 in correct order. You do not need to copy the code, just to indicate which part of the code goes in each of 1 - 5 slots.

```
int main() {
    int fd, clientfd;
    socklen_t len;
    struct sockaddr_in r, q;
    char buf[80];
```

1

```
    memset(&r, '\0', sizeof (r));
    r.sin_family = AF_INET;
    r.sin_addr.s_addr = INADDR_ANY;
    r.sin_port = htons(1234);
```

2

3

```
    len = sizeof (q);
```

4

5

```
    buf[len] = '\0';
    close(clientfd);
    return(0);
}
```

4.5. [3 points]. The following program intercepts *Ctrl+C* signal and prints a message *"Got signal!"*, without terminating the program. Fill-in missing variables and function parameters.

```
void my_handler(int signum) {
  const char msg[] = "Got signal\n";
  write (STDOUT_FILENO, msg, sizeof (msg));
}

int main(int argc, char *argv[]) {
  printf("PID: %d\n", getpid());

  // Set up signal handler
  struct sigaction action;
  action.sa_handler =_______________;
  sigaction (________, _______, NULL);

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

Part 5. Concepts [8 points]

Mark the following statements as *true* or *false*. If *false*, explain why. If *true*, do not explain.

5.1. After we call *free (p)* on any pointer variable *p*, the value of *p* becomes *NULL*.

5.2. Variables *a* and *pa* are declared as following:

```
char *pa = "mango";
char a [] = "mango";
```

There is no difference between *a* and *pa*. They can be used interchangeably in any context.

5.3. Sorting nodes of a linked list is useful, because we would be able to find an element of interest using binary search.

5.4. Given a parent and a child process, a single pipe establishes a communication in one direction only - either from parent to child or from child to parent - and cannot be used to transfer data in both directions.

Part 6. [10 points]. Concepts in code

Consider a **32-bit** system where *int* takes 4 bytes, *long* takes 8 bytes, *char* takes 1 byte and *address* takes 4 bytes.

For each piece of code below, say what is printed.

Code | **Output**

6.1.

```
int main() {
   char str1[] = "mystring";
   char str2[] = {'m','y','s','t','r','i','n','g'};
   int n1 = sizeof(str1)/sizeof(str1[0]);
   int n2 = sizeof(str2)/sizeof(str2[0]);
   printf("%d %d", n1, n2);
   return 0;
}
```

6.2.

```
void print_numbers (char numbers[]) {
  for (int i = 0; i< sizeof (numbers); i++) {
         printf("%c", numbers[i]);
    }
}
int main () {
    char a[] = "123456789";
    print_numbers (a);
    return 0;
}
```

6.3.

```
void fun(int y) {
    y = 55;
}
int main() {
  int y = 44;
  fun(y);
  printf("%d", y);
  return 0;
}
```

6.4.

```
int * ip;
long * lp;
printf ("%ld%ld", sizeof(ip), sizeof(lp));
```

6.5.

```
typedef struct {
        char a;
        long n;
}N9;
printf ("%ld", sizeof(N9));
```

Part 7. [10 points]. Memory segments

[2 points]. Draw a memory diagram of a single process (either horizontally or vertically). Name all the memory segments in correct order.

```
char  x[] = "ab";

void func() {

        char c = 'a';

   A.   char *p = &c;
   B.   p = x;
   C.   p = "cdf";
   D.   p = malloc (5* sizeof (char) );
   E.   free (p);

}
```

In the above code, variable *p* is declared on the stack.

[8 points]. For each line of code marked by A-E, indicate to what memory segment *p* is pointing to, by drawing an arrow between each letter (A,B,C,D,E) and the corresponding memory segment.

Appendix

For question 4.4. Code pieces to match:

A
```
if (listen(fd, 5)) {
      perror("listen");
      return(1);
}
```

B
```
if ((fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("socket");
        return(1);
}
```

C
```
if ((len = read (clientfd, buf, sizeof buf - 1)) < 0) {
        perror("read");
        return(1);
}
```

D
```
if ((clientfd = accept(fd, (struct sockaddr *)&q, &len)) < 0) {
        perror("accept");
        return(1);
 }
```

E
```
if (bind(fd, (struct sockaddr *)&r, sizeof r) < 0) {
        perror("bind");
        return(1);
}
```

Useful functions, structs and constants

Standard C library

```
void* calloc (size_t num, size_t size);
void* malloc (size_t size);
void qsort(void *base, size_t buff, size_t size,
                  int (*compar)(const void *, const void *))
long strtol (const char *restrict_str, char **restrict_endptr, int base);
```

I/O

```
int fclose (FILE *stream)
char *fgets (char *s, int n, FILE *stream)
FILE *fopen (const char *file, const char *mode)
int fprintf (FILE * stream, const char * restrict_format, ...);
ssize_t fread (void *ptr, size_t size, size_t nmemb, FILE *stream);
int fseek (FILE *stream, long offset, int whence);
      /* whence = SEEK_SET, SEEK_CUR, or SEEK_END */
ssize_t fwrite (const void *ptr, size_t size, size_t nmemb, FILE *stream);
int sprintf (char *s, const char *format, ...)
```

Strings

```
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)
```

File descriptors

```
int close (int fd)
int dup2 (int oldfd, int newfd)
int pipe (int filedes[2])
ssize_t read (int d, void *buf, size_t nbytes);
int open (const char *path, int oflag)
      /* oflag is O_WRONLY | O_CREAT for write, O_RDONLY for read */
int fileno (FILE *stream)
ssize_t write(int d, const void *buf, size_t nbytes);
```

System calls and processes

```
int execl (const char *path, const char *arg0, const char *arg1 , …, NULL);
int execvp (const char *file, char *argv[])
pid_t fork (void)
pid_t getpid (void);
pid_t getppid (void);
```

Inter-process communication

```
int wait (int *status)
      WIFEXITED(status) WEXITSTATUS(status)
      WIFSIGNALED(status) WTERMSIG(status)
      WIFSTOPPED(status) WSTOPSIG(status)

int kill (int pid, int signo)
int sigaction (int signum,
               const struct sigaction *act,
               struct sigaction *oldact)
      /* signum = SIGINT, SIGQUIT, SIGKILL, SIGTERM etc.*/
      struct sigaction:
            void (*sa handler)(int);
            sigset_t sa mask;
            int sa_flags;
int sigaddset (sigset_t *set, int signum)
int sigemptyset (sigset_t *set)
int sigprocmask (int how, const sigset_t *set, sigset_t *oldset)
      /*how can be SIG_BLOCK, SIG_UNBLOCK, or SIG_SETMASK */

int accept (int sock, struct sockaddr *addr, int *addrlen)
int bind (int sock, struct sockaddr *addr, int addrlen)
int connect (int sock, struct sockaddr *addr, int addrlen)
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)
unsigned long int htonl (unsigned long int hostlong) /* 4 bytes */
unsigned short int htons (unsigned short int hostshort) /* 2 bytes */
int listen (int sock, int n)
```

```
unsigned long int ntohl (unsigned long int netlong)
unsigned short int ntohs (unsigned short int netshort)
int select (int maxfdp1, fd_set *readfds, fd_set *writefds,
            fd_set *exceptfds, struct timeval *timeout)
int socket (int family, int type, int protocol)
      /* family=PF INET, type=SOCK STREAM, protocol=0 */
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*/
```

Shell scripting

Comparison operators

`-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

Shell commands

`cat, cut, echo, ls, sort, uniq`

`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

`-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

Positional parameters

`$0` Script name

`$#` Number of positional parameters

`$*` List of all positional parameters

`$?` Exit value of previously executed command