1. Write shell commands to perform the following operations. Put some thought into making your answer simple; answers will be graded on quality, not just on whether they "work".
   Please be careful to write single quotes and backquotes correctly to avoid misinterpretation.

   (a) Rename every file in the current directory whose name begins with a q so as to double the q. For example, "question" would be renamed to "qquestion".

```
for file in $(ls | grep "^q") ; do
    mv "$file" "q$file"
    done
```

   (b) The file /u/ajr/t contains zero or more integers, one per line. Output the sum of all of these integers plus 10. (For example, if the file is empty, the output would be 10, or if it contains "`2\n3\n`" the output would be 15.)

```
sum=0
for number in $(cat /u/ajr/t) ; do
    sum=$(expr $sum + $number)
    done
echo $(expr $sum + 10)
```

   (c) The file /u/ajr/x contains a sample of ordinary English text. Output the total number of letters in this file. (That is, your answer differs from the output of "`wc </u/ajr/x`" because wc also counts digits, punctuation, newlines, etc.)

```
cat "/u/ajr/x" | tr -c -d [:alpha:] | wc -m
# Deletes everything that is not a letter from file, and then uses wc
```

2. You meant to type "`grep fred bigfile`", but you accidentally typed "`grep fred`" and pressed return (enter). What will happen? What exactly is grep doing?
   `grep` will search `stdin` for matches to "fred"

3. A file named "pricelist" in the current directory contains item names and prices, looking like this:

```
pop 150
sandwich 500
french fries 225
```

The prices are all in cents, so that all numbers are positive integers. The item and price are always separated by a space, but the item may contain spaces in its name, so the separator is the *last* space on the line. (There cannot be a space after the price.)

(a) Write shell commands to output the list of all item names which occur more than once in this file. (Any such duplication is an error; these shell commands check for this error.)

```
list=
for item in $( cat pricelist | tr " " / ) ; do
    list="$list$(dirname $item | tr / " " )\n"
    done
echo $list | sort | uniq -d
```

(b) Write a shell script which takes two mandatory command-line arguments, where the first is an item name and the second is a quantity. The output of the shell script is the total price (in cents). For example, with the above file, "sh script pop 2" would output 300. You can assume that the item does not occur more than once in the file (i.e. the part 'a' error check has already been done before putting this pricelist file into production), but you can't assume that the item occurs; you need to check for this and give an error message if applicable.

```
pass=""
for item in $( cat pricelist | tr " " / ) ; do
    if [ "$1" = "$(dirname $item | tr / " " )" ] ; then
        echo $(expr $(basename $item) \* $2)
        pass=y
        break
    fi
    done
[ -z $pass ] && echo "item not found"
```

4. Write a complete C program which takes no command-line arguments (you don't have to declare the argc and argv arguments to main()) and functions like "`tr a-z A-Z`" – that is, stdin is copied to stdout with all letters capitalized and all other characters copied unmodified.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MAX_STR 256

int main()
{
    char buf[MAX_STR];
    if (fgets(buf, MAX_STR, stdin) == NULL) {
        perror("fgets");
        return -1;
    }
    for (int i = 0; i < strlen(buf); i++) {
        printf("%c", isupper(buf[i]) ? tolower(buf[i]) : buf[i]);
    }
    return 0;
}
```

5. Write a complete C program which takes two mandatory command-line arguments. The first is a string and the second is an integer. The string is output that many times.
   Example, where '$' is the shell prompt:

```
$ ./a.out hello 3
hello
hello
hello
$
```

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    if (argc != 3) {
        fprintf(stderr, "Incorrect number of arguments\n");
        return -1;
    }
    int times;
    if ((times = atoi(argv[2])) == -1) {
        perror("atoi");
        return -1;
    }
    for (int i = 0; i < times; i++) {
        printf("%s\n", argv[1]);
    }
    return 0;
}
```

6. Write a complete C program which attempts to fopen() files named “1”, “2”, and so on up to “100” inclusive. When the fopen() succeeds, it checks whether the first character in the file is a ‘#’. When the fopen() fails, it just loops around to the next file without emitting any error message (i.e. it’s ok for these files not to exist). At the end of your program’s execution, it outputs the total count of files (out of those examined) which do indeed have an ‘#’ as their first character.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
    FILE *file;
    char buf[10];
    char hashtest[2];
    int sum = 0;
    for (int i = 1; i < 101; i++) {
        sprintf(buf, "%d", i);
        if ((file = fopen(buf, "r")) == NULL) {
            continue;
        }
        if (fgets(hashtest, 2, file) == NULL) {
            fclose(file);
            continue;
        }
        if (*hashtest == '#') {
            sum++;
        }
        fclose(file);
    }
    printf("%d\n", sum);
}
```

7. Write a complete C program which takes a mandatory directory name argument as argv[1]. It reads the directory (non-recursively) and creates a linked list of the names in it in the following data structure (the items can be in any order):

```
struct file {
        char *name;
        struct file *next;
};
```

Do not include "." and ".." in the linked list.
The name is simply the base file name in the directory; you don't have to prepend the path name. Space for the name must be malloc'd. After doing the above, we would presumably go on to *use* this data structure, but for the purposes of this exam, your program will simply exit.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <dirent.h>

struct file {
    char *name;
    struct file *next;
};

void printlist(struct file *head) {
    for (struct file *temp = head; temp != NULL; temp = temp->next) {
        printf("%s -> ", temp->name);
    }
    printf("\n");
}

struct file *create(char *name) {
    struct file *node = (struct file *) calloc(1, sizeof(struct file));
    node->name = (char *) calloc(strlen(name) + 1, sizeof(char));
    strcpy(node->name, name);
    node->next = NULL;
    return node;
}

struct file *add(struct file *node, struct file *head) {
    if (head == NULL) {
        return node;
    }
    node->next = head;
    return node;
}

int main(int argc, char *argv[])
{
    if (argc != 2) {
        fprintf(stderr, "Incorrect number of arguments\n");
    }
    if (chdir(argv[1]) == -1) {
        perror("chdir");
        return -1;
    }
    DIR *folder;
    folder = opendir(".");
    struct dirent *entry;
    struct file *head = NULL;
    if (folder == NULL) {
        fprintf(stderr, "Unable to read directory\n");
        return -1;
```

```
    }
    while ((entry = readdir(folder))) {
        if ((strcmp(entry->d_name, ".") == 0) || (strcmp(entry->d_name, "..") == 0)) {
            continue;
        }
        head = add(create(entry->d_name), head);
    }
    // printlist(head);
    return 0;
}
```

8. (a) In the unix filesystem, when you create a new file, its mtime and ctime are initialized to the current time. What do you think the atime should be set to? Propose the rule for the initialization of the atime and justify your decision extremely briefly.

   atime should be set to the current time as well, since the os "accesses" the file when creating it.

   (b) What is the link count on a new file? (Keep in mind that a directory is a kind of file.)

   A new file has one link, while a new directory has an extra link which is '..' in its directory.

9. The program named "/u/ajr/crazy" does something useful, but causes problems when run automatically because it might read from stdin and/or write to stdout or stderr.

   We want to suppress the i/o behaviour by connecting all of stdin, stdout, and stderr to /dev/null.

   Write a complete C program which executes /u/ajr/crazy with these redirections in place. (All appropriate error checking is required.)

```c
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <stdlib.h>
#include <stdio.h>

int main()
{
    int null;
    if ((null = open("/dev/null", O_RDWR)) == -1) {
        perror("open");
        return -1;
    }
    if (dup2(null, STDOUT_FILENO) == -1) {
        perror("dup2");
        return -1;
    }
    close(STDOUT_FILENO);
    if (dup2(null, STDIN_FILENO) == -1) {
        perror("dup2");
        return -1;
    }
    close(STDIN_FILENO);
    if (dup2(null, STDERR_FILENO) == -1) {
        return -1;
    }
    close(STDERR_FILENO);
    if (execl("/u/ajr/crazy", "/u/ajr/crazy", (char *) NULL) == -1) {
        exit(1);
    }
}
```

10. There is a computationally-intensive process which has been separated into two independent pieces, implemented by the functions f() and g(), each of which takes no parameters and returns an integer.

    Write a complete C program which calls fork; one process computes f() and the other computes g(); and then your program outputs the sum of the two values. You will need a pipe so that the child process can convey the result to the parent.
    (There's no exec(); all of this happens within one program.)

```c
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <stdlib.h>
#include <stdio.h>

int f() {
    return 69;
}

int g() {
    return 420;
}

int main()
{
    int chpipe[2];
    if (pipe(chpipe) == -1) {
        perror("pipe");
        return -1;
    }
    switch (fork())
    {
        case -1:
            perror("fork");
            exit(1);
            break;
        case 0:
            close(chpipe[0]); // Will not be reading from pipe in child
            int fret_val = f();
            write(chpipe[1], &fret_val, sizeof(fret_val));
            break;
        default:
            close(chpipe[1]); // Will not be writing to pipe in parent
            int gret_val = g();
            int ffret_val;
            read(chpipe[0], &ffret_val, sizeof(fret_val));
            printf("%d\n", ffret_val+gret_val);
            break;
    }
    return 0;
}
```

11. The following program is a server which reads one byte (character) from each client, echoes it back, and drops the connection.

```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main()
{
    int fd, clientfd;
    socklen_t len;
    struct sockaddr_in r, q;
    char c;

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

    memset(&r, '\0', sizeof r);
    r.sin_family = AF_INET;
    r.sin_addr.s_addr = INADDR_ANY;
    r.sin_port = htons(2000);

    if (bind(fd, (struct sockaddr *)&r, sizeof r) < 0) {
        perror("bind");
        return(1);
    }
    if (listen(fd, 5)) {
        perror("listen");
        return(1);
    }

    while (1) {
        len = sizeof q;
        if ((clientfd = accept(fd, (struct sockaddr *)&q, &len)) < 0) {
            perror ("accept" );
            return(1);
        }
        switch (read(clientfd, &c, 1)) {
            case -1:
                perror("read");
                return(1);
            case 1:
                if (write(clientfd, &c, 1) != 1)
                    perror("write");
            }
            close(clientfd);
    }
}
```

(a) Regarding the line "r.sin_port = htons(2000);" (line 23), what would happen if instead we simply wrote "r.sin_port = 2000;"?
The network protocols used need to have the port number stored in network order, which htons() does for us, if 2000 was used alone, the port number would be stored in host order which would cause networking problems.

(b) On which line number will this program be blocked when no client is connected?
36

(c) On which line number will this program be blocked when a client is connected but has not yet transmitted its byte?
40

(d) In the switch statement beginning on line 40, under what circumstances will neither of the cases match? What will read()'s return value be, and why?
When connection is broken from client's side and no data is available, read will return 0 and neither of the cases will match.

(e) Change the program in place so that it mostly functions as it currently does, except that if the byte transmitted from the client is a control-B (ASCII value 2), instead of sending the control-B back it sends the string "ha!" plus a network newline.

```
1  // Changed code beginning at line 14
2
3      while (1) {
4          len = sizeof q;
5          if ((clientfd = accept(fd, (struct sockaddr *)&q, &len)) < 0) {
6              perror ("accept" );
7              return(1);
8          }
9          switch (read(clientfd, &c, 1)) {
10             case -1:
11                 perror("read");
12                 return(1);
13             case 1:
14                 if (c == 2) {
15                     char *ha = "ha!\n"
16                     if (write(clientfd, ha, 4) != 4) {
17                         perror("write");
18                     }
19                 }
20                 if ((c != 2) && (write(clientfd, &c, 1) != 1))
21                     perror("write");
22             }
23             close(clientfd);
24     }
25 }
```