CiTR Unix Skills Test

This test contains 18 questions, some of which are multi-part. Questions are in multiple choice form, but there may be more than one correct answer to the questions. Tick (✔) all the answers that you think are correct. Incorrect answers lose marks, correct answers gain marks and no answer gives no change. Not all questions have a correct answer listed.

Question 1: platforms and binary compatibility

In the following, it is assumed that

- on Solaris, Sun supplied compilers are used, and
- on Linux that GNU compilers are used
- on Windows that GNU compilers are used

For each of the following, which could reasonably be expected to work for applications that are not attempting platform specific behaviors?

1 ❒ x.sh written under Sun Solaris on SPARC, run on Linux on Intel

2 ❒ x.o compiled on Sun/Solaris on Sparc, linked with main.o compiled on Sun/Solaris on Intel, and if successful, run on Sun/Solaris on Intel

3 ❒ x compiled and linked on Linux on Intel, run on Windows on Intel

4 ❒ x compiled and linked on Sun/Solaris on Sparc, run on Sun/Solaris on Intel

Question 2: Commands

Unix has many obscure commands. For each of the commands listed below, mark the ones for which the description is reasonable.

1	❒ `at`	displays the ASCII table
2	❒ `cat`	joins files together
3	❒ `cmd`	executes a new command shell
4	❒ `comm`	finds common lines in input file
5	❒ `creat`	Makes a new file
6	❒ `dirs`	lists working directory stack
7	❒ `ksh`	executes a new command shell
8	❒ `lpq`	will list files queued to the printer
9	❒ `lpstat`	will list files queued to the printer
10	❒ `ls`	lists files in folder/directory
11	❒ `mv`	renames a file
12	❒ `pr`	sends file to default printer
13	❒ `ps`	sends file to default postscript printer
14	❒ `rm`	deletes a file
15	❒ `tee`	makes a second copy of input file
16	❒ `uniq`	identifies duplicated files in a directory
17	❒ `wc`	display who is logged onto console
18	❒ `write`	sets the standard output filename
19	❒ `xargs`	displays arguments used starting X server
20	❒ `xdpyinfo`	displays screen size

Question 3: directories/file naming

3.1 What is "`..`" in the file system?

1 ❒ all of the current subdirectories
2 ❒ current root directory
3 ❒ current working directory
4 ❒ definition of a range of filenames
5 ❒ device mounted in the filesystem
6 ❒ parent directory

3.2 Which of the following characters are legal in a unix file system's filename

1 ❒ Any alphanumeric character
2 ❒ Blank/space character
3 ❒ Back slash character "\"
4 ❒ Colon character ":"
5 ❒ Forward slash character "/"

3.3 Which of the following is true about filenames in Unix

1 ❒ a period "." at the beginning of a filename indicates a directory
2 ❒ a period "." at the beginning of a filename indicates a hidden file
3 ❒ a period "." designates file type and is hence not stored as part of the name
4 ❒ a filename ending in ".a" is typically an object library
5 ❒ a filename ending in ".a" is typically an assembly language file
6 ❒ a filename ending in ".dll" is typically a shareable library
7 ❒ a filename ending in ".lib" is typically an object library
8 ❒ a filename ending in ".s" is typically an assembly language file
9 ❒ a filename ending in ".so" is typically a shareable library
10 ❒ a filename ending in ".so" is typically a StarOffice document

Question 4: File names

A directory contains 6 files with the following names. For each of the regular expressions below, which files are identified through the shell?

```
FReda.h
Wilma.h
Wilmah.h
[bB]etty.c
freda.c
petty.h
```

4.1 `petty.c`

1. ❒ `FReda.h`
2. ❒ `Wilma.h`
3. ❒ `Wilmah.h`
4. ❒ `[bB]etty.c`
5. ❒ `freda.c`
6. ❒ `petty.h`

4.2 `*.c`

1. ❒ `FReda.h`
2. ❒ `Wilma.h`
3. ❒ `Wilmah.h`
4. ❒ `[bB]etty.c`
5. ❒ `freda.c`
6. ❒ `petty.h`

4.3 `*a.*`

1. ❒ `FReda.h`
2. ❒ `Wilma.h`
3. ❒ `Wilmah.h`
4. ❒ `[bB]etty.c`
5. ❒ `freda.c`
6. ❒ `petty.h`

4.4 [fF][rR]eda.*

1. ❒ FReda.h
2. ❒ Wilma.h
3. ❒ Wilmah.h
4. ❒ [bB]etty.c
5. ❒ freda.c
6. ❒ petty.h

4.5 [bB]etty.*

1. ❒ FReda.h
2. ❒ Wilma.h
3. ❒ Wilmah.h
4. ❒ [bB]etty.c
5. ❒ freda.c
6. ❒ petty.h

4.6 [a-zA-Z]etty.*

1. ❒ FReda.h
2. ❒ Wilma.h
3. ❒ Wilmah.h
4. ❒ [bB]etty.c
5. ❒ freda.c
6. ❒ petty.h

Question 5: Editors

A text file contains a comma separated list of name and phone numbers with one person per line.

Wilma,Flintstone,+1-342-237729
Fred,Flintstone,+1-342-237729
Fred,Nurk,+81-202-005670
Eva,Nurk,+81-202-005670
Barney,Rubble,+1 (303) 5678901
Fred,Rubble,+1 (303) 5678901
Homer,Simpson,+46 21 334678
Ren,Stimpy,+45 22 446688

The following conventions are used for keystrokes.

↵	carriage return
③	function key 3
❹	Alt key 4
¶	escape

The task is to change the third occurrence of "Fred" to "Betty". You are at the top of the file. Which of the following keystroke sequences will achieve the goal. (An optimal solution is not required).

1 ❒ using vi

```
/Fred↵nncwBetty¶
```

2 ❒ using vi

```
3s/Fred/Betty/↵
```

3 ❒ using notepad

```
③Fred↵↵↵❹Betty
```

4 ❒ using ex

```
/Fred↵/↵/↵s//Betty/p↵
```

Question 6: filters

A text file named “phone.csv” contains a comma separated list of name and phone numbers with one person per line.

Wilma,Flintstone,+1-342-237729
Fred,Flintstone,+1-342-237729
Fred,Nurk,+81-202-005670
Eva,Nurk,+81-202-005670
Barney,Rubble,+1 (303) 5678901
Fred,Rubble,+1 (303) 5678901
Homer,Simpson,+46 21 334678
Ren,Stimpy,+45 22 446688

The task is to create an new file called “phone.pwd” with all the commas(,) changed to colons(:). Which of the following shell scripts will achieve the result. (An optimal solution is not required).

1 ❒ # using strsub

```
strsub ',' ':' < phone.csv > phone.pwd
```

2 ❒ # using tr

```
cat phone.csv | tr ',' ':' > phone.pwd
```

3 ❒ # using grep

```
grep 's/,/:/g' phone.csv > phone.pwd
```

4 ❒ # using ed

```
cat < phone.csv > phone.pwd
ed phone.pwd << EOF
1,\$s/,/:/g
w
q
EOF
```

Question 7: File permissions

The directory *freddo* has been created by user *bloggs* as shown by below. You are logged in as user *stimpy*, and *freddo* is your current working directory. The output of the `id` command when you run it is:

```
uid=987(stimpy) gid=1103(student)
```

The output of the `id` command when *bloggs* runs it is:

```
uid=370(bloggs) gid=10(staff)
```

Directory freddo

```
drwx----w-   2 bloggs   staff         69 Sep 15 18:42 assign
drwxr-x--x   2 bloggs   staff        107 Sep 15 18:42 dira
-rw-r-----   1 bloggs   staff         29 Sep 15 18:42 filea
-rw-rw-r--   1 bloggs   staff         29 Sep 15 18:42 fileb
-rwxrwxrwx   1 bloggs   staff         29 Sep 15 18:42 filec
-r-xr-xr-x   1 bloggs   staff       5888 Sep 15 18:42 id1
-r-sr-xr-x   1 bloggs   staff       5888 Sep 15 18:42 id2
-r-xr-sr-x   1 bloggs   staff       5888 Sep 15 18:42 id3
```

Directory of subdirectory dira

```
-rw-rw-r--   1 bloggs   staff         29 Sep 15 18:42 filec
```

For this exercise, the files whose name starts with '*file*' are the output of the `date` command and the files whose name starts with "*id*" are copies of the `id` program.

7.1 What is displayed when you enter the command

```
more filea
```

1 ❒ `filea`
2 ❒ `filea: No such file or directory`
3 ❒ `filea: Permission denied`
4 ❒ `Fri Sep 15 18:42:07 EST 2000`

7.2 What is displayed when you enter the command

```
more dira/filec
```

1 ❒ `filec`
2 ❒ `filec: No such file or directory`
3 ❒ `filec: Permission denied`
4 ❒ `Fri Sep 15 18:42:07 EST 2000`

7.3 What would be the intended purpose of the directory *assign*?

1 ❒ Place for a student group to collaborate on a project
2 ❒ Place for instructor to provide templates for students to fill in
3 ❒ Place for instructor to hide answers to a test
4 ❒ Place for students to submit assignment answers to a test

7.4 What is displayed when you enter the command

```
./id2
```

1 ❒ `Fri Sep 15 18:42:07 EST 2000`
2 ❒ `id2`
3 ❒ `id2: Permission denied`
4 ❒ `uid=987(stimpy) gid=1103(student)`
5 ❒ `uid=987(stimpy) gid=1103(student) euid=370(bloggs)`
6 ❒ `uid=987(stimpy) gid=1103(student) euid=987(stimpy)`

7.5 You change your working directory to */tmp*. Here you execute the command

```
date > now
```

which yields the directory entry

```
-rw-rw-r--  1 stimpy  student 29 Sep 15 19:58 now
```

You issue another command

```
chmod 600 now
```

which yields the directory entry

```
-rw-------  1 stimpy  student 29 Sep 15 19:58 now
```

In order to achieve the same file permission for *now*, what command could you substitute for *COMMAND* in the following command sequence :

```
COMMAND
date > now
```

1 ❒ `chmod 600 now`
2 ❒ `umask 600`
3 ❒ `umask 600 now`
4 ❒ `umask 066`
5 ❒ `umask 066 now`

Question 8: File duplication

For each DUPLICATE command (8.1…8.5) substituted in the command sequence below, what is displayed at RESULT ?

```
date > a
                        <=== DUPLICATE
rm a
more b
                        <=== RESULT
```

8.1 `copy a b`

1 ❒ current date and time is displayed

2 ❒ error message “no such file or directory”

8.2 `ln a b`

1 ❒ current date and time is displayed

2 ❒ error message “no such file or directory”

8.3 `ln -s a b`

1 ❒ current date and time is displayed

2 ❒ error message “no such file or directory”

8.4 `cat a b`

1 ❒ current date and time is displayed

2 ❒ error message “no such file or directory”

8.5 `cp a b`

1 ❒ current date and time is displayed

2 ❒ error message “no such file or directory”

Question 9: kill

For each of the kill commands below, indicate which of comments about the command are true. In these questions it is assumed that the user issuing the kill command has sufficient privileges.

9.1 `kill -9` *process_id*

1 ❒ lowers priority of process while system releases resources
2 ❒ memory dump is left
3 ❒ parent process is not sent an indication of process completion
4 ❒ process is given a chance to clean up before exit
5 ❒ process will always exit

9.2 `kill -0` *process_id*

1 ❒ forces truncation of the memory dump file
2 ❒ indicates whether process still exists
3 ❒ process and any child processes are forced to exit
4 ❒ process will always exit

9.3 `kill -SIGHUP` *process_id*

1 ❒ ends a virtual machine session
2 ❒ force a memory dump, without exiting process
3 ❒ requests process to re-read configuration setups
4 ❒ signify a processor halt, and process migration

Question 10: X

You are logged onto server *brute*. There is a workstation with a single console screen and keyboard called *arch*. *Arch* supports an X server and has access control turned off. A second server also exists, called *wimp*.

For each of the tasks below which command sequence(s) will achieve the desired result?

10.1 Display a clock with the current time of *brute* on *arch*.

1 ❒
```
xclock -display=brute
```

2 ❒
```
xclock -display arch:0.0
```

3 ❒
```
xclock -xhost=arch
```

4 ❒
```
xhost brute xclock=arch:0.0
```

10.2 On workstation *arch*, display a login window for *wimp*

1 ❒
```
DISPLAY=arch:0.0; export DISPLAY
xterm -e telnet wimp &
```

2 ❒
```
XHOST=arch; export XHOST
xterm -e telnet wimp &
```

3 ❒
```
XHOST=arch; export XHOST
telnet wimp -e xterm &
```

4 ❒
```
xhost wimp xterm=arch:0
```

5 ❒
```
xterm -display arch:0 -e telnet wimp
```

6 ❒
```
xterm -xhost=arch -e telnet wimp
```

Question 11: compiling, linking, running

A directory contains (only) the following files

a.c

```
int fa(void) { return 10; }
```

b.c

```
static int eleven = 11;
static int fb(int b){
    return (b + eleven);
}
```

c.c

```
int eleven = 11;
int fb(int b){
    return (b + eleven);
}
```

m.c

```
#include <stdio.h>
extern int fa();
extern int fb(int);
int main() {
    int x = fb(fa());
    printf("fb(fa())=%d\n", x);
}
```

n.c

```
#include <stdio.h>
extern int eleven();
extern int fb(int);
int main() {
    int x = fb(eleven());
    printf("fb(eleven())=%d\n", x);
}
```

A series of commands is run as setup before the questions below.

```
rm core a.out.core *.o *.a
cc -c a.c
cc -c b.c
cc -c c.c
```

For each question below, you would stop executing the commands after the first fatal error, so if a compilation fails then don't bother reporting that running the executable fails.

For each question, what happens when the command is run?

11.1 `ar -crv libocal.a a.o b.o c.o`

1 ❒ ar gives error "no such file or directory"
2 ❒ ar gives error "duplicate symbol"
3 ❒ ar gives no error
4 ❒ cc gives compilation error
5 ❒ cc gives error "no such file or directory"
6 ❒ cc gives error "duplicate symbol"
7 ❒ cc gives error "missing symbol"
8 ❒ a.out gives error "core dumped"
9 ❒ a.out produces output "fb(fa())=%d"
10 ❒ a.out produces output "fb(fa())=11"
11 ❒ a.out produces output "fb(fa())=21"
12 ❒ a.out produces no output

11.2
```
cc m.c a.o b.o
./a.out
```

1 ❒ ar gives error "no such file or directory"
2 ❒ ar gives error "duplicate symbol"
3 ❒ ar gives no error
4 ❒ cc gives compilation error
5 ❒ cc gives error "no such file or directory"
6 ❒ cc gives error "duplicate symbol"
7 ❒ cc gives error "missing symbol"
8 ❒ a.out gives error "core dumped"
9 ❒ a.out produces output "fb(fa())=%d"
10 ❒ a.out produces output "fb(fa())=11"
11 ❒ a.out produces output "fb(fa())=21"
12 ❒ a.out produces no output

11.3

```
cc m.c a.o c.o
./a.out
```

1 ❒ ar gives error “no such file or directory”
2 ❒ ar gives error “duplicate symbol”
3 ❒ ar gives no error
4 ❒ cc gives compilation error
5 ❒ cc gives error “no such file or directory”
6 ❒ cc gives error “duplicate symbol”
7 ❒ cc gives error “missing symbol”
8 ❒ a.out gives error “core dumped”
9 ❒ a.out produces output “fb(fa())=%d”
10 ❒ a.out produces output “fb(fa())=11”
11 ❒ a.out produces output “fb(fa())=21”
12 ❒ a.out produces no output

11.4

```
ar -crv libocal.a a.o c.o
cc m.c -L. -local
./a.out
```

1 ❒ ar gives error “no such file or directory”
2 ❒ ar gives error “duplicate symbol”
3 ❒ ar gives no error
4 ❒ cc gives compilation error
5 ❒ cc gives error “no such file or directory”
6 ❒ cc gives error “duplicate symbol”
7 ❒ cc gives error “missing symbol”
8 ❒ a.out gives error “core dumped”
9 ❒ a.out produces output “fb(fa())=%d”
10 ❒ a.out produces output “fb(fa())=11”
11 ❒ a.out produces output “fb(fa())=21”
12 ❒ a.out produces no output

11.5

```
ar -crv libocal.a a.o c.o
cc m.c a.o c.o -L. -local
./a.out
```

1 ❒ ar gives error “no such file or directory”
2 ❒ ar gives error “duplicate symbol”
3 ❒ ar gives no error
4 ❒ cc gives compilation error
5 ❒ cc gives error “no such file or directory”
6 ❒ cc gives error “duplicate symbol”
7 ❒ cc gives error “missing symbol”
8 ❒ a.out gives error “core dumped”
9 ❒ a.out produces output “fb(fa())=%d”
10 ❒ a.out produces output “fb(fa())=11”
11 ❒ a.out produces output “fb(fa())=21”
12 ❒ a.out produces no output

11.6

```
ar -crv libocal.a a.o c.o
cc m.c -L. -local a.o c.o
./a.out
```

1 ❒ ar gives error “no such file or directory”
2 ❒ ar gives error “duplicate symbol”
3 ❒ ar gives no error
4 ❒ cc gives compilation error
5 ❒ cc gives error “no such file or directory”
6 ❒ cc gives error “duplicate symbol”
7 ❒ cc gives error “missing symbol”
8 ❒ a.out gives error “core dumped”
9 ❒ a.out produces output “fb(fa())=%d”
10 ❒ a.out produces output “fb(fa())=11”
11 ❒ a.out produces output “fb(fa())=21”
12 ❒ a.out produces no output

11.7

```
ar -crv libocal.a a.o c.o
cc n.c -L. -local
./a.out
```

1 ❒ ar gives error “no such file or directory”
2 ❒ ar gives error “duplicate symbol”
3 ❒ ar gives no error
4 ❒ cc gives compilation error
5 ❒ cc gives error “no such file or directory”
6 ❒ cc gives error “duplicate symbol”
7 ❒ cc gives error “missing symbol”
8 ❒ a.out gives error “core dumped”
9 ❒ a.out produces output “fb(fa())=%d”
10 ❒ a.out produces output “fb(fa())=11”
11 ❒ a.out produces output “fb(fa())=21”
12 ❒ a.out produces no output

Question 12: Libraries and system calls

Below are a number of macros, library and system calls that may be used when programming on Unix. For each of the calls listed below, mark the ones for which the description is reasonable.

1 ❒ `calloc` is used to request initialized memory from the process heap

2 ❒ `chroot` is used to restrict the subtree of the file system a process sees

3 ❒ `exec` creates a new Unix process, and runs the given command

4 ❒ FD_ISSET can be used to determine when to read data from a remote computer

5 ❒ `fcntl` can be used to set file locking

6 ❒ `fopen` is used to read Unix directory files

7 ❒ `fork` is used to create a new directory entry for a file

8 ❒ `free` is used to remove a file from a directory

9 ❒ `gethostbyaddr` reads machine location from installation log

10 ❒ `kill` sends a message code to another Unix process

11 ❒ `mkdir` is used to create a new directory

12 ❒ `nice` is used by a process to change its running priority

13 ❒ `open` is used to establish a handle to a file for subsequent reading

14 ❒ `fprintf` formats and prints arguments using a handle from `open`

15 ❒ `select` can be used to determine when to read data from a remote computer

16 ❒ `signal` sends a message code to another Unix process

17 ❒ `socket` can be used to establish process to process communication

18 ❒ `stat` can be used to find which user owns a file

19 ❒ `system` creates a new Unix process, and runs the given command

20 ❒ `unlink` is used to remove a file from a directory

Question 13: System utilities

13.1 CD

In Bourne shell, “cd” is built into the shell rather than implemented as a stand-alone utility. The compelling reason for this is:

1 ❒ because the CD drive is viewed through the file system
2 ❒ efficiency concerns
3 ❒ faster as filesystem searching can be bypassed
4 ❒ lack of a corresponding system call
5 ❒ it allows a history/audit trail to be maintained
6 ❒ it can only affect the currently running process

13.2 Other special commands

Which of the following shell commands cannot be implemented as stand-alone utilities?

1 ❒ `.`
2 ❒ `echo`
3 ❒ `exec`
4 ❒ `find`
5 ❒ `kill`
6 ❒ `printenv`
7 ❒ `time`
8 ❒ `ulimit`
9 ❒ `umask`

Question 14: shared libraries

The directory *Lib* contains

Directory Lib

```
libX.a
libX.so
X.so
libb.a
```

The executable *testme* is compiled using :-

```
cc -o testme main.c -L Lib -lX -lb
```

14.1 Which of the following will *testme* have as a shared object?

1 ❒ libX.a
2 ❒ libX.so
3 ❒ X.so
4 ❒ libb.a

14.2 Which of the following will *testme* have as a static object?

1 ❒ libX.a
2 ❒ libX.so
3 ❒ X.so
4 ❒ libb.a

14.3 A change is made to a library, and you had to relink *testme* to incorporate the change. Identify which libraries would cause this situation?

1 ❒ libX.a
2 ❒ libX.so
3 ❒ X.so
4 ❒ libb.a

14.4 Which library will definitely cause a link error if any symbol remains unresolved?

1 ❒ libX.a
2 ❒ libX.so
3 ❒ X.so
4 ❒ libb.a

Question 15: Software development tools

For each of the software development tasks below identify the tool/toolset that best assists.

15.1 Automate a software build

1 ❒ adb
2 ❒ ar
3 ❒ builtin
4 ❒ cmp
5 ❒ df
6 ❒ du
7 ❒ ldd
8 ❒ link
9 ❒ lint
10 ❒ lookbib
11 ❒ make
12 ❒ nm
13 ❒ psrinfo
14 ❒ RCS
15 ❒ tar
16 ❒ tr

15.2 Coordinate file editing on a multi-developer/multi-file project

1 ❒ adb
2 ❒ ar
3 ❒ builtin
4 ❒ cmp
5 ❒ df
6 ❒ du
7 ❒ ldd
8 ❒ link
9 ❒ lint
10 ❒ lookbib
11 ❒ make
12 ❒ nm
13 ❒ psrinfo
14 ❒ RCS
15 ❒ tar
16 ❒ tr

15.3 Determine overall disk space utilization

1 ❒ adb
2 ❒ ar
3 ❒ builtin
4 ❒ cmp
5 ❒ df
6 ❒ du
7 ❒ ldd
8 ❒ link
9 ❒ lint
10 ❒ lookbib
11 ❒ make
12 ❒ nm
13 ❒ psrinfo
14 ❒ RCS
15 ❒ tar
16 ❒ tr

15.4 Determine which symbols are referenced and exported by an object file

1 ❒ adb
2 ❒ ar
3 ❒ builtin
4 ❒ cmp
5 ❒ df
6 ❒ du
7 ❒ ldd
8 ❒ link
9 ❒ lint
10 ❒ lookbib
11 ❒ make
12 ❒ nm
13 ❒ psrinfo
14 ❒ RCS
15 ❒ tar
16 ❒ tr

15.5 Distributing a snapshot of a source tree to another office

1 ❒ adb
2 ❒ ar
3 ❒ builtin
4 ❒ cmp
5 ❒ df
6 ❒ du
7 ❒ ldd
8 ❒ link
9 ❒ lint
10 ❒ lookbib
11 ❒ make
12 ❒ nm
13 ❒ psrinfo
14 ❒ RCS
15 ❒ tar
16 ❒ tr

15.6 Examine which shared libraries are used by an executable

1 ❒ adb
2 ❒ ar
3 ❒ builtin
4 ❒ cmp
5 ❒ df
6 ❒ du
7 ❒ ldd
8 ❒ link
9 ❒ lint
10 ❒ lookbib
11 ❒ make
12 ❒ nm
13 ❒ psrinfo
14 ❒ RCS
15 ❒ tar
16 ❒ tr

15.7 Regression test of trace output

1 ❒ adb
2 ❒ ar
3 ❒ builtin
4 ❒ cmp
5 ❒ df
6 ❒ du
7 ❒ ldd
8 ❒ link
9 ❒ lint
10 ❒ lookbib
11 ❒ make
12 ❒ nm
13 ❒ psrinfo
14 ❒ RCS
15 ❒ tar
16 ❒ tr

15.8 Show a call traceback at the time of a program crash from its dump

1 ❒ adb
2 ❒ ar
3 ❒ builtin
4 ❒ cmp
5 ❒ df
6 ❒ du
7 ❒ ldd
8 ❒ link
9 ❒ lint
10 ❒ lookbib
11 ❒ make
12 ❒ nm
13 ❒ psrinfo
14 ❒ RCS
15 ❒ tar
16 ❒ tr

Question 16: make

Makefile

```
FRED=fred.c

wilma: wilma.c
    cc -I. -o wilma wilma.c

fred: $(FRED)
    cc -I. -o fred $(FRED)

flintstones: wilma fred

wilma.c: flintstone.h

$(FRED): flintstone.h

clean:
    rm Makefile wilma.c $(FRED) flintstone.h
    rm fred wilma
```

A directory contains the file *Makefile* as shown above and the files *flintstone.h, wilma.c* and *fred.c*. What shell commands are executed if you type `make`?

1 ❒ `FRED=fred.c`
2 ❒ `wilma: wilma.c`
3 ❒ `cc -I. -o wilma wilma.c`
4 ❒ `fred: $(FRED)`
5 ❒ `fred: fred.c`
6 ❒ `cc -I. -o fred $(FRED)`
7 ❒ `cc -I. -o fred fred.c`
8 ❒ `flintstones: wilma fred`
9 ❒ `wilma.c: flintstone.h`
10 ❒ `$(FRED): flintstone.h`
11 ❒ `fred.c: flintstone.h`
12 ❒ `clean:`
13 ❒ `rm Makefile wilma.c $(FRED) flintstone.h`
14 ❒ `rm Makefile wilma.c fred.c flintstone.h`
15 ❒ `rm fred wilma`

Question 17: Unix documentation

17.1 Unix `man` pages are prepared as

1 ❒ Latex files
2 ❒ Plain text files
3 ❒ Rich text files
4 ❒ Nroff files
5 ❒ StarOffice documents
6 ❒ HTML files
7 ❒ Info files
8 ❒ Postscript files

17.2 Unix man pages are normally found in

1 ❒ /usr/man
2 ❒ /opt/man
3 ❒ /usr/doc
4 ❒ /etc/man
5 ❒ /usr/local/emacs/manuals
6 ❒ /usr/share/man

Question 18: Unix services and daemons

Below are a number of services to be found on Unix. For each of the calls listed below, mark the ones for which the description is reasonable.

1	❒ `at`	executes recurrent user commands at specific times
2	❒ `cron`	executes recurrent user commands at specific times
3	❒ `DNS`	supports the digital certificate repository
4	❒ `inetd`	starts IP protocol services
5	❒ `init`	adopts processes whose parent process has died
6	❒ `init`	starts specific other Unix services
7	❒ `nfs`	supports the network time synchronization service
8	❒ `NIS`	includes a network wide login security database
9	❒ `rlogin`	provides restricted login
10	❒ `tftpd`	services the trivial file transfer protocol
11	❒ `vmstat`	displays operation of the Intel x86 emulation service
12	❒ `YP`	includes a network wide login security database