Duration: **50 minutes**
Aids Allowed: **None**

Student Number: _ _ _ _ _ _ _ _ _

Last Name: SOLUTION

First Name: _____

TA: _____ **Instructor:** Reid

---

*Do **not** turn this page until you have received the signal to start.*
(In the meantime, please fill out the identification section above, and read the instructions below *carefully*.)

This midterm test consists of 5 questions on 6 pages (including this one), plus the aid sheet. *When you receive the signal to start, please make sure that your copy of the test is complete.* Extra space was left for each of the programming questions. Please indicate clearly the part of your work that should be marked.

IMPORTANT: You do not need to include the "#!" line in Bourne shell programs you are asked to write. In C programs, you do not need to add "#include" lines, or do error checking unless the question requires it, or the program would not function correctly given valid input without error checking.

MARKING GUIDE

1: _____/ 4

2: _____/ 9

3: _____/ 5

4: _____/ 8

5: _____/ 7

TOTAL: _____/33

Good Luck!

Question 1. [4 MARKS]

Part (a) [1 MARK]

What is the output of the following sequence of shell commands? The sequence is given in tcsh and bash syntax, but the output is the same in both cases.

```
# tcsh syntax
setenv PATH /usr/bin:/usr/local/bin:.
setenv PATH /u/fred/bin:$PATH
echo $PATH
```

```
# bash or sh syntax
export PATH=/usr/bin:/usr/local/bin:.
export PATH=/u/fred/bin:$PATH
echo $PATH
```

```
/u/fred/bin:/usr/bin:/usr/local/bin:.
```

Part (b) [1 MARK]

A student sent me the following question about assignment 2: "I have tried to run my c program in cygwin. I can compile them, but when I type '`pathlist /path1/path2`' it gives me 'bash : pathlist : command not found.' " What is the most likely problem?

The current working directory is not in the PATH.

Part (c) [2 MARKS]

Rewrite the following shell program so that it does not use any temporary files, and the output is the same. The program finds the total grade in each student's marks file, extracts each mark and prints the average of all of the marks.

```
grep TOTAL */Marks.txt > temp1
cut -d " " -f 2 < temp1 > temp2
python avg.py <temp2 > temp3
cat temp3
rm temp1 temp2 temp3
```

```
grep TOTAL */Marks.txt | cut -d " " -f 2 | python avg.py
```


CONT'D...

Question 2. [9 MARKS]

Part (a) [4 MARKS]

The current working directory contains an executable shell program called `doit` which is shown below. Write the output produced by the following `echo` commands and the contents of the file `outfile` after each command has executed. Assume that `outfile` is empty before each command is executed. (' is a single quote, ` is a backquote.)

```
#!/bin/sh
# doit
echo "$1"
```

	prints	**contents of outfile**
`echo doit now > outfile`	nothing	`doit now`
`echo "doit now > outfile"`	`doit now > outfile`	`nothing`
`echo ‘doit now‘ > outfile`	nothing	` now`
`echo ‘doit now > outfile‘`	blank line	`now`

Part (b) [5 MARKS]

Write a Bourne shell program that prints the size and the name of each file in the current working directory that is bigger than `size` bytes, where `size` is a command-line argument to the shell program. The program must not print out directories. Recall that `stat -t filename` prints a line of information about the file, `filename`. The second field of this line gives the size of the file in bytes. The command `cut -d " " -f 2` may be used to extract the second field from a line separated by spaces. Do not use `ls` or `cat`.

```
for file in *
do
    if [ -f $file ]
    then
        size=‘stat -t $file | cut -d " " -f 2‘
        if [ "$size" -gt "$1" ]
        then
            echo "$size  $file"
        fi
    fi
done
```


CONT'D...

Question 3. [5 MARKS]

Consider the following program. In your answers below, assume that all processes terminate normally.

```
int main() {

  int p1 = fork();
  int p2 = fork();

  if(p1 == 0) {
    printf("A\n");

    if(p2 == 0) {
      printf("B\n");
    }
  }
  printf("C\n");
  return(0);
}
```

Part (a) [1 MARK]

How many processes are created? 4

Part (b) [1 MARK]

How many times is "C" printed? 4

Part (c) [1 MARK]

How many times is "B" printed? 1

Part (d) [1 MARK]

How many times is "A" printed? 2

Part (e) [1 MARK]

Is it possible for "A" to be printed after "B"? Yes

CONT'D...

Question 4. [8 MARKS]

Part (a) [5 MARKS]

Consider the following C declarations. Write the type of the expressions that follow. If the type is `char` then give the value of the character.

```
char **a = malloc(3 * sizeof(char *));
a[0] = "bcd";
a[1] = "efg";
a[2] = "hij";
```

	Type	Value (if type is char)
`a[0]`	pointer to char	_____
`a[0][1]`	char	c
`*(a+1)`	char	e
`*a[1]`	char	3
`&a[2]`	pointer to pointer to char	_____

Part (b) [3 MARKS]

Complete the C function below that returns the number of occurrences of the character `c` in the string `str`.

```
int countchars(char *str, char c) {
  char *ptr = str;
  int count = 0;
  while( (ptr = strchr(ptr, c)) != NULL) {
    count++;
    ptr++;
  }
  return count;
}

int countchars2(char *str, char c) {
  int count = 0;
  int i;
  for(i = 0; i < strlen(str); i++) {
    if(str[i] == c) {
      count++;
    }
  }
  return count;
}
```


CONT'D...

Question 5. [7 MARKS]

Write a C program that takes a file as a command line argument. The file contains a list of file names, one per line. Your program prints the name of each file in the list that is **not** in the current working directory. Your program should work correctly even if there are hundreds of files in the current working directory and hundreds of file names in the list. You may assume that the file is well-formatted. Each line will contain only a potential file name.

```
int
main(int argc, char **argv)
{
  FILE *fp, *tp;
  char line[256];
  if(argc != 2) {
    fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
    exit(1);
  }

  if((fp = fopen(argv[1], "r")) == NULL) {
    perror("fopen");
    exit(1);
  }

  while(fgets(line, 256, fp) != NULL) {
    line[strlen(line)-1] = '\0';
    if((tp = fopen(line, "r")) == NULL) {
      printf("%s\n", line);
    } else {
      fclose(tp);
    }
  }

  return 0;
}
```

Notes:

Another easy way to do this was to use `stat` rather than `fopen` to test for the existence of the file.

Many students tried to solve this problem using `readdir`. There is a correct solution, but it is a bit tricky.

END OF SOLUTIONS