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 7 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:	______/ 6
# 2:	______/ 7
# 3:	______/ 5
# 4:	______/ 9
# 5:	______/ 6
TOTAL:	______/33

Good Luck!

Question 1. [6 MARKS]

Part (a) [2 MARKS]

Briefly explain what the PATH environment variable contains, and what it is used for.

It stores a list of directories. This list is used to locate executable files, rather than specifying the absolute path.

Part (b) [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 var is set as shown below and outfile is empty before each command is executed. (’ is a single quote, ‘ is a backquote.)

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

	prints	**contents of outfile**
var="then"		
doit "$var" > outfile	nothing	then
doit ’$var’ > outfile	nothing	$var
echo ‘doit $var > outfile‘	blank line	then
echo doit "now > outfile"	doit now > outfile	nothing

Question 2. [7 MARKS]

Write a Bourne shell program that counts the number of files (not directories) in each of the **subdirectories** of the current working directory. You do not need to worry about hidden directories or files. Do not use `ls` or `cat`. (You should not need the whole page to write the program.)

```
for f in *
do
    if [ -d $f ]
    then
        count=0
        for file in $f/*
        do
            if [ -f $file ]
            then
                count=`expr $count + 1`
            fi
        done
        echo $f $count
    fi
done
```

Question 3. [5 MARKS]

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

```
int main() {
  int p1, p2;

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

    p2 = fork();
    if(p2 == 0) {
      sleep(2);
      printf("B\n");
      exit(0);
    }
    wait(0);
  }

  printf("C\n");
  return(0);
}
```

Part (a) [1 MARK]

How many processes are created (including the original process)? 3

Part (b) [1 MARK]

How many times is "C" printed? 2

Part (c) [1 MARK]

How many times is "A" printed? 1

Part (d) [1 MARK]

Is it possible for a "C" to be printed before an "A"? Yes

Part (e) [1 MARK]

Is it possible for "B" to be printed last? No

Question 4. [9 MARKS]

Parts a) and b) refer to the following C statements:

```
    char *p1, *p2;
    char *a = malloc(10 * sizeof(char));
    strncpy(a, "bcdefghij", 10);
    p1 = a;  p2 = a;
```

Part (a) [1 MARK]

Write a C program fragment using pointer arithmetic that sets `p1` to point to the character `'d'` in the character array `a`. Do not use array subscripts.

```
    while(*p1 != 'd')
       p1++;
```

or simply

```
p1 += 2;
```

Part (b) [1 MARK]

Write a C program fragment using array subscripts that sets `p2` to point to the character `'g'` in the character array `a`. Do not use pointer arithmetic.

```
    for(i = 0; i < strlen(a); i++)
      if(p2[i] == 'g') {
          p2 = &p2[i];
          break;
      }
```

or

```
p2 = a[5];
```

Part (c) [3 MARKS]

Complete the C function below.

```
/* Returns a pointer to the first occurrence of the character c in the
 * string s. Returns NULL if the character is not found.*/

char *my_strchr(char *s, char c) {

    int i;
    for(i = 0; i < strlen(str); i++) {
        if(str[i] == c) {
            return &str[i];
        }
    }
    return NULL;
}
```

Part (d) [4 MARKS]

There are three errors in the code below that tests the function in part c). The code compiles cleanly, without warnings. Identify and describe **two** of the errors and explain how to fix them.

```
int main()
{
    char *strings[3] = {"Fun", "with", "pointers"};
    char *p = malloc(strlen("pointers")+1);
    int i;

    for(i = 0; i < sizeof(strings); i++) {
        p = my_strchr(strings[i], 'n');
        printf("p now points to %c\n", *p);
    }
    return 0;
}
```

- memory leak in malloc for p – don't need it
- Using sizeof is incorrect. It will give us 12, not 3. Use 3 instead.
- If `my_strchr` returns NULL then we will get a seg fault. Check that p is not NULL before printing.

Question 5. [6 MARKS]

Write a C program that takes zero or more command line arguments. You program will check each argument and print a message indicating whether the argument is a file, a directory, or neither a valid file nor a directory.

For example suppose the current working directory contains a file called `file1` and a directory called `dir1` and the program is called `checkargs`. There is also a directory `/tmp`, and `/dev/null` exists, but is neither a file nor a directory. `notafile` does not exist.

Then `checkargs file1 dir1 notafile /dev/null /tmp` will print

```
file1 is a file
dir1 is a directory
notafile is not a file or a directory
/dev/null is not a file or a directory
/tmp is a directory
```

```
int
main(int argc, char **argv)
{
  int i;
  struct stat sbuf;
  for(i = 1; i < argc; i++) {

    if(stat(argv[i], &sbuf) != -1 ) {
      if(S_ISREG(sbuf.st_mode)) {
    printf("%s is a file\n", argv[i]);
      } else if (S_ISDIR(sbuf.st_mode)) {
    printf("%s is a directory\n", argv[i]);
      } else {
    printf("%s is not a valid file or directory\n", argv[i]);
      }
    } else {
    printf("%s is not a valid file or directory\n", argv[i]);
    }
  }
  return 0;
}
```