Duration: **50 minutes**
Aids Allowed: **1 - 8.5x11 sheet**

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 9 pages (including this one). *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: _____/ 6

3: _____/ 5

4: _____/ 6

5: _____/ 8

TOTAL: _____/31

Good Luck!

Question 1. [6 MARKS]

Part (a) [3 MARKS]

Suppose the current working directory contains the following two directories:

```
d-wx-wx-wx     me   d1
drw-rw-rw-     me   d2
```

and the permissions on the files in those directories are as follows:

```
-rwxr-xr-x     me   d1/xfile
-rwxr-xr-x     me   d2/yfile
```

Circle the statements below that will produce an error.

[`ls d1`]	`cd d1`	`d1/xfile`
`ls d2`	[`cd d2`]	[`d2/yfile`]

Part (b) [2 MARKS]

Write 2 lines of C code that would result in a memory leak.

```
char *s = malloc(10);
s = ''Hello'';
```

Part (c) [1 MARK]

Rewrite your example in part c, fixing the error but using exactly the same variable(s).

```
char *s;
s = ``Hello'';
```

Question 2. [6 MARKS]

Suppose you have a program called `concat` that

- reads from standard input
- concatenates all of the lines you read into one long string
- prints that string to standard output
- returns the number of lines it read.

Part (a) [3 MARKS]

Write a shell program that uses `concat` to print the number of lines in the file `testfile`. The program should **not** print anything else. (Do not use any other program to determine the number of lines in `testfile`.)

```
concat < testfile > /dev/null 2>&1
echo $?
```

Part (b) [3 MARKS]

Suppose the file `filelist` contains a list of file names. Write a single shell command that uses `concat` and makes one call to `grep` so that it searches for the word "penguin" in the the files given in `filelist`.

```
grep penguin `concat < filelist`
```

Question 3. [5 MARKS]

A web administrator wants to make sure that the files on the site are not writable. Write a shell program called `iswritable` that prints out the **absolute path** (including the file name) of each writable file in and below the current working directory.

```
#!/bin/sh

for file in *
do
    if [ -d $file ]
    then
        cd $file
        iswritable
        cd ..
    else
        if [ -w $file ]
        then
            echo $PWD/$file
        fi
    fi
done
```

Question 4. [6 MARKS]

Write a C function that takes a string as an argument, and prints each word of the string on a separate line. Two words are separated by one or more spaces. Your function may modify the string that is passed in, but it may not allocate any more memory for strings, and you may not print one character at a time.

```
void splitprint(char *str)
{
  char *startptr = str;
  char *endptr;

  while((endptr = strchr(startptr, ' ')) != NULL) {
    *endptr = '\0';
    printf("%s\n", startptr);
    endptr++;
    startptr = endptr;
    while(*startptr == ' ') {
      startptr++;
    }
  }
  printf("%s\n", startptr);
}
```

Using strtok is okay even though that makes the question pretty simple.

Question 5. [8 MARKS]

Complete the following C program to validate an archive file from assignment 2. The program takes an archive file as a command line argument, and prints an error message for each file in the archive that does not appear in the current working directory. It will also print an error message if the size of the file differs from the metadata information stored in the archive file.

Assume you are given the following function. Do not implement `readMetadata`

```
/* The readMetadata function takes two argument: a file pointer to the archive file and a
 * pointer to an array of record structs as its second.  It reads the metadata from the
 * archive and stores the information in the array of structs. The function returns the
 * number of lines of metadata that it read.
 */
int readMetadata(FILE *fp, struct record *md);

struct record {
    char *name;
    int size;
    mode_t mode;
};

int
main(int argc, char **argv)
{
    struct record metadata[MAXFILES];
```

```
  int i;
  int numfiles = 0;
  struct stat sbuf;
  FILE *infp;

  if(argc != 2) {
      fprintf(stderr, "Usage: unarchive <filename>\n");
      exit(1);
  }

  if((infp = fopen(argv[1], "r")) == NULL) {
    fprintf(stderr, "Could not open %s. Terminating.\n", argv[1]);
    exit(1);
  }

  numfiles = readMetadata(infp, metadata);
  fclose();

  for(i = 0; i < numfiles; i++) {
    if(stat(metadata[i].name, &sbuf) == -1) {
      printf("Error: %s not found\n", metadata[i].name);
    } else {
```

```
      if(sbuf.st_size != metadata[i].size) {
        printf("Error: %s is %d but the metatdata reports it as %d\n",
               metadata[i].name, (int)sbuf.st_size, metadata[i].size);
      }
    }
  }
  return 0;
}
```

(This page intentionally left blank)

C functions for strings:

```
char *index(const char *s, int c);
char *strncat(char *dest, const char *src, size_t n);
char *strchr(const char *s, int c);
size_t strlen(const char *s);
int strncmp(const char *s1, const char *s2, size_t n);
char *strncpy(char *dest, const char *src, size_t n);
char *strstr(const char *haystack, const char *needle);
```

C functions for files and directories:

```
int closedir(DIR *dir);
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 *format, ...);
char *getcwd(char *buf, size_t size);
DIR *opendir(const char *name);
struct dirent *readdir(DIR *dir);
int stat(const char *file_name, struct stat *buf);
void perror(const char *s);
```

```
struct stat {
  dev_t         st_dev;      /* device */
  ino_t         st_ino;      /* inode */
  mode_t        st_mode;     /* protection */
  nlink_t       st_nlink;    /* number of hard links */
  off_t         st_size;     /* total size, in bytes */
  unsigned long st_blksize;  /* blocksize for filesystem I/O */
  unsigned long st_blocks;   /* number of blocks allocated */
  time_t        st_atime;    /* time of last access */
  time_t        st_mtime;    /* time of last modification */
  time_t        st_ctime;    /* time of last change */
};
```

```
The following POSIX macro functions are defined to check the file type
(m is the st_mode field of the stat struct):
S_ISLNK(m)  is it a symbolic link?
S_ISREG(m)  regular file?
S_ISDIR(m)  directory?
```

Shell comparison operators

Shell	Description
`-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.