Question 1. [6 MARKS]

All parts of this question assume the following C statement. Parts (b) through (e) assume a variable called `ptrs`.

```
char data[256] = "Hop Pop We like to hop.";
```

Part (a) [1 MARK] Is the following statement allowed? Why or why not?

```
data[0] = 'P';
```

Part (b) [1 MARK]

We want to use a variable `ptrs` to point to the first character of each word in `data`. Fill in the argument to `malloc` so that enough memory is allocated to store a pointer to the first character of each word in `data`.

```
char **ptrs = malloc(6 * sizeof(char*));
```

Part (c) [1 MARK]

Using array notation, write a statement that makes the first element in `ptrs` refer to the first character of the first word in `data`.

```
ptrs[0] = &data[0];
```

Part (d) [1 MARK]

Using only pointer notation, write a statement that makes the second element in `ptrs` refer to the first character of the second word in data without changing the value of `ptrs`.

```
*(ptrs+1) = data + 4;
```

Part (e) [2 MARKS] Write two C statements so that the following statements prints "like".

```
ptrs[0] = &data[11];
data[15] = '\0';



printf("%s\n", *ptrs);
```

Question 2. [6 MARKS]

Write a shell program that takes a path to a directory as command-line argument, and prints to standard output the sum of the sizes of the regular files in the directory. It counts only the sizes of files and not directories or other special devices.

Use the command `du` to get the size of a file. For example, assuming the current working directory contains a file called test.pdf, `du test.pdf` prints to standard output the line

```
185       test.pdf
```

(There are other options to `du` that will do almost what this question asks. You must not use them.)

```
total=0

for file in $1/*
do
    if [ -f $file ]
    then
        size=‘du $file | cut -f 1‘
        echo $size
        total=‘expr $total + $size‘
    fi
done
echo $total
```

Question 3. [3 MARKS]

In assignment 1, the entire metadata array was read from the file and written to the file. Another possibility would have been to read or write just one element of the array.

Write a snippet of C code that reads the 4th `Fnode` from the open `FILE` pointer `fp`. You will need to declare the appropriate variable to store the data that is read from the file. Assume the current position of the file is the first byte of the file, and that the data exists in the file. No error handling is required.

Recall that the `Fnode` struct was defined as follows:

```
typedef struct file_metadata {
        char name[MAXNAME];
        int offset;
        int length;
} Fnode;

Fnode f;
fseek(fp, 4*sizeof(Fnode), SEEK_SET);
fread(&f, sizeof(Fnode), fp);
```

Question 4. [6 MARKS]

Part (a) [5 MARKS]

The current working directory contains two files shown below with their contents.

cmd1

```
wc -w
```

cmd2

```
*1
```

Recall that `wc -w FILE` prints the number of words in FILE followed by name of FILE.

Two variables are defined below. For each of the following lines, give the output that would appear when the command is run. (‘ is a backquote and ’ is a forward quote)

```
x="cat"
y='cat cmd1 cmd2'
```

```
echo $x *          => cat cmd1 cmd2

echo "$x *"        => cat *

echo `$x *`        => wc -w cmd1

echo '$y'          => $y

echo `$y`          => wc -w cmd1
```

Part (b) [1 MARK]

What are the minimum permissions needed on `cmd1` and `cmd2` for all of the commands in part (a) to run without permission errors.

Read permissions.

Question 5. [7 MARKS]

Part (a) [6 MARKS]

Complete the C function below that takes the string `line` as an argument and removes the html tags from the line. It returns a pointer to the beginning of the string.

An html tag is defined as all characters that lie between < and > including the < and > symbols themselves. You may assume well-formed data. In other words, there will be a closing > for every <.

The function must make the changes to the string **in place**. You may **not** declare another array of characters or use `malloc`.

```
char *remove_tags(char* line) {

        int r = 0;
        int s = 0;
        int in_tag = 0;
        while(line[s] != '\0') {
            if(in_tag) {
                if(line[s] == '>') {
                    in_tag = 0;
                }
                s++;

            } else {
                if(line[s] == '<') {
                    in_tag = 1;
                    s++;
                } else {
                    line[r] = line[s];
                    r++;
                    s++;
                }
            }
        }
        line[r] = '\0';
        return line;
    }
```

Part (b) [1 MARK] If the function above did not return a pointer to the resulting string, would the changes to the string be visible to the function that called `remove_tags`? Explain briefly.

x

C function prototypes and structs:

```
int fclose(FILE *stream)
char *fgets(char *s, int n, FILE *stream)
FILE *fopen(const char *file, const char *mode)
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
int fseek(FILE *stream, long offset, int whence)
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
char *index(const char *s, int c)
DIR *opendir(const char *name)
void perror(const char *s)
struct dirent *readdir(DIR *dir)
unsigned int sleep(unsigned int seconds)
int sprintf(char *s, const char *format, ...)
int stat(const char *file_name, struct stat *buf)
char *strchr(const char *s, int c)
size_t strlen(const char *s)
char *strncat(char *dest, const char *src, size_t n)
int strncmp(const char *s1, const char *s2, size_t n)
char *strncpy(char *dest, const char *src, size_t n)
char *strrchr(const char *s, int c)
char *strstr(const char *haystack, const char *needle)
```

Useful structs

```
struct stat { /*NOTE: only fields that might be needed are included */
    mode_t st_mode; /* inode protection mode */
    uid_t st_uid; /* user-id of owner */
    gid_t st_gid; /* group-id of owner */
    off_t st_size; /* file size, in bytes */
};
```

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.

Useful Unix programs for shell programs: `cat, cut, wc, grep, sort, sort -n (for numerical sorting), head, tail`