Question 1. [4 MARKS]

Part (a) [2 MARKS]

I asked the TAs to run a program called `get_submissions` that is stored in `/u/csc209h/bin`.

Write the command to run `get_submissions` using an absolute path.

```
/u/csc209h/bin/get_submissions
```

When a TA tried to run the program they received the message "permission denied". Explain precisely what might have caused this. *The user does not have execute permission on the the file get_submission, or may not have execute permissions on some directotry in the path.*

Part (b) [2 MARKS]

In assignment 2, `packetize` and `readstream` took arguments as follows:

```
    packetize [-f inputfile] outputfile
    readstream [-l logfile] inputfile
```

`packetize` reads from `inputfile` if it is provided as an argument, or from standard input otherwise. `packetize` writes packets to `outputfile`.

`readstream` reads packets from `inputfile` and writes its output to standard output. (The log file isn't relevant for this question.)

The file `packets.b` contains packets in the appropriate format to be input for `readstream`.

Write one line in the shell that will take the data produced by `readstream` from `packets.b` and create a new list of packets (using `packetize`) in a new file, `uniform.b`. No new files may be created.

```
readstream packets.b | packetize uniform.b
```

Question 2. [4 MARKS]

An `unsigned short` can be used to store a set of numbers between 0 and 15. The value `k` is in the set if the bit at index `k` is 1.

Complete the function `printset` below that prints the numbers in `set`.

For example, given the following call to `printset`,

```
    unsigned short myset = 0b1000100010001001;
    printset(myset);
```

the output is

```
0
3
7
11
15
```

```
    void printset(unsigned short set) {

    void printset(unsigned short set) {
        int i;
        for(i=0; i < 16; i++) {
            if((set >> i ) & 1)
                printf("%d\n", i);
        }
    }
```

Question 3. [12 MARKS]

Consider the following code. Please read through the code and the answer questions below.

```
#define MAXSIZE 256

struct rink {
    char *name[4];
    int score;
};

void set_score(int *score, int value) {
    *score = value;
}

int main() {
    struct rink *can = malloc(sizeof(struct rink));

    char buffer[256];
    int i;
    for(i = 0; i < 4; i++) {
        fgets(buffer, MAXSIZE, stdin);
        // A)


        add_player(can, i, buffer);
    }

    set_score(&can->score, 10);

    // B)

    for(i = 0; i < 4; i++) {
        // We print i+1 because team positions really start at 1 not 0
        printf("%d %s\n", i+1, can->name[i]);
    }
    printf("Score: %d\n", can->score);

    cleanup(can);
    return 0;
}
```

Part (a) [1 MARK]

How many bytes of memory are allocated in the line below? ____________________

```
struct rink *can = malloc(sizeof(struct rink));
```

*4*sizeof(char *) + 4 = 20 or 36 (or 40 since padding to a word boundary on a 64-bit machine gives 40 bytes)*

Part (b) [1 MARK]

Write one line of code after the comment `A` that removes the newline character from `buffer`.

```
buffer[strlen(buffer)-1] = '\0';
```

Part (c) [3 MARKS]

Write the function `add_player` so that it can be called as shown in `main`. The second argument is the index into the `name` array of a `struct rink`.

```
    void add_player(struct rink *r, int pos, char *str) {
        r->name[pos] = malloc(strlen(str) + 1);
        strncpy(r->name[pos], str, strlen(str) + 1);
    }
```

Part (d) [2 MARKS]

Write the function `cleanup` to free memory appropriately.

```
    void cleanup(struct rink *r) {

    void cleanup(struct rink *r) {
        int i;
        for(i = 0; i <4; i++) {
            free(r->name[i];)
        }
        free(r);
    }
```

Part (e) [1 MARK]

The file "`TeamCanada`" contains the following

```
    Askin
    Officer
    Lawes
    Jones
```

Assuming the code above is compiled to produce an executable called `maketeam`, write the command to call it so that the input comes from the file `TeamCanada`.

```
maketeam < TeamCanada
```

Part (f) [2 MARKS]

Suppose we wanted to change one of the team members in `can`, and wrote the following line after comment B in the above code.

```
can->name[0] = "Arnott";
```

The correct output is produced, but the program that operated correctly before this line was added has a segmentation fault. What is the problem?

You can't call free on a string literal

Part (g) [2 MARKS]

Suppose we use a different approach to change one of the team members, and wrote the following line after comment B in the above code (instead of the line in Part (f)).

```
add_player(can, 0, "Arnott");
```

Now the program runs without any errors, but valgrind reports that some memory hasn't been freed. What is the problem?

Memory leak if add_player doesn't free the memory first.

Question 4. [5 marks]

```
int main() {

    int r = fork();
    printf("B\n");

    int t = fork();
    printf("C\n");

    if(r == 0) {
        printf("D\n");
    } else if (r > 0 && t == 0){
        printf("E\n");
    }
    printf("F\n");
    return 0;
}
```

Part (a) [1 mark]

How many processes are created, including the first process to execute `main`?

4

Part (b) [1 mark] Is it possible for a `B` to be printed after a `C`?

Part (c) [1 mark] How many times is `D` printed?

Part (d) [2 marks]

Describe an order in which the processes could run such that a process would become a orphan. (Be clear about which process is the orphan and what has to happen for that process to become a orphan.)

Example: If the first parent runs quickly and terminates before the first child terminates (before the first child prints F), then the first child would be an orphan.