Question 1. [4 marks]

Part (a) [2 marks]

Suppose the current working directory is `/tmp`, and contains an executable file called `runtests`. Write a command that executes `runtests` using its absolute path.

```
/tmp/runtests
```

Write a command that executes `runtests` using a relative path.

```
./runtests (or runtests)
```

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.)

Explain the changes that would need to be made to `packetize` and `readstream` so that the following command would run correctly. (We want to send the packets created by `packetize` directly to `readstream`.)

```
packetize -f inputfile | readstream -l logfile > output
```

packetize would have to write to standard output, and readstream would have to read from standard input.

Question 2. [4 MARKS]

In the function check defined below, the argument status has two different pieces of information packed into it. The upper 8 bits of status hold a value, and the lower 8 bits hold flags.

Complete the code below so that:

- flag is 0 if the bit at index 2 is 0, and non-zero if the bit at index 2 is 1
- value is set to the value of the upper 8 bits of status

For example, if status is 0001001000001111 then the printf statement will print "value is 18", but if status was 0001001000001011, then the printf statement will print "flag is not set"

```
void check(unsigned short status) {

    //int flag = status & 0x02;
    int flag = status >> 8;


    if(flag) {

        int value =  (status & 0xff00) >> 8;

        printf("value is %d", value);

    } else {

        printf("flag is not set\n");

    }
}
```

Question 3. [12 MARKS]

Please read through the following code and the questions on the next page first.

```
struct player {
    char *name;
    int goals;
};

struct player *create_player(char *n, int g) {

    // A) Fill in the argument to malloc

    struct player *p = malloc(sizeof(struct player));

    // B)


    p->name = malloc(strlen(n) + 1);
    strncpy(p->name, n, strlen(n) + 1);
    p->goals = g;


    return p;
}

void score(struct player p) {
    p.goals += 1;
}

struct player **init_roster(int size) {

    struct player **team = malloc(size * sizeof(struct player *));

    int i;
    for(i = 0; i < size; i++ ) {
        team[i] = NULL;
    }
    return team;
}
```

```
int main() {
    struct player **team = init_roster(20);

    team[0] = create_player("Agosta-Marciano", 3);
    team[1] = create_player("Poulin", 3);
    struct player p = {"Johnston", 2};
    team[2] = &p;
    p.name = "Wickenheiser";
    p.goals = 2;
    team[3] = &p;

    score(*team[1]);

    int i = 0;
    while(team[i] != NULL) {
        printf("%s %d\n", team[i]->name, team[i]->goals);
        i++;
    }
}
```

Part (a) [1 MARK]

Fill in the argument to `malloc` after comment A in the code.

Part (b) [3 MARKS]

Write the code after comment B that will make a copy of `n`, assign it to the `name` field of `p`, and assign the `goals` field of `p`.

Part (c) [4 MARKS]

Write the output of this program.

```
Agosta-Marciano 3
Poulin 3
Wickenheiser 2
Wickenheiser 2
```

Part (d) [4 MARKS]

Complete the function cleanup to free all the memory allocated for team. Assume create_player was used to create all players in the team, and that if team[i] == NULL, it marks the end of the list.

```
void cleanup(struct player **team) {
```

```
  void cleanup(struct player **team) {
    int i = 0;
    while(team[i] != NULL) {
      free(team[i]->name);
      free(team[i]);
   i++;
    }
    free(team);
  }
```

Question 4. [5 marks]

```
int main() {

    int r = fork();

    if(r == 0) {
        printf("C\n");

    } else {
        r = fork();
        printf("D\n");
        if(r == 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`?

3

Part (b) [1 mark]

Can an `E` be printed before a `C`?

Yes

Part (c) [1 mark]

How many times is `D` printed?

2

Part (d) [2 marks]

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

If the parent proces is the last to terminate, the first child will be a zombie