## Question 1. [8 MARKS]

The following program reads star data from a text file in the following format. Each line contains a string (with no spaces in it) representing the name of a star followed by a space and the magnitude of the star as a floating point number. In writing the code below, you may assume that the file has the correct format.

Example:

```
Polaris 1.97
Vega 0.03
Aldebaran 0.87
```

### Part (a) [5 MARKS]

Complete the function `read_stars` that will read star data from the open text file pointer passed in as an argument. It will create and build a linked list of `struct star`'s from this data.

- New elements may be inserted at the head of the list.
- If the file is empty, return NULL
- Assume the length of a star name is no more than `MAXSIZE - 1`.
- Values stored in the linked list should occupy no more space than necesssary.

```
struct star {
    char *name;
    double magnitude;
    struct star *next;
};

struct star *read_stars(File *fp)  {

    char str[MAXSIZE];
    struct star *head = NULL;
    double mag;

    while(fscanf(fp, "%s %lf", str, &mag) != EOF) {
        struct star *cur = malloc(sizeof(struct star));
        cur->name = malloc(strlen(str) + 1);
        strncpy(cur->name, str, strlen(str) + 1);
        cur->magnitude = mag;
        cur->next = head;
        head = cur;
    }
    return head;
}
```

**Part (b)** [3 MARKS]

Complete the function below that calls free on memory allocated for the linked list of stars, `head`. Remember not to leave dangling pointers.

```
void free_stars(struct star *head) {

    while(head != NULL) {
        struct star *tmp;
        free(head->name);
        tmp = head;
        head = head->next;
        free(tmp);
    }
}
```

## Question 2. [2 MARKS]

I spent a long time working on my `print_ftree` program on cdf and it works perfectly, but when I compile and run it on another Linux machine, once in a while it gives me a segmentation fault.

**Part (a)** [1 MARK] Is there a bug in my code? Explain your answer.

*Yes! The bug may not be noticed because the memory that was being used incorrectly coincidently had the correct values or was coincidentally not used in the first case.*

**Part (b)** [1 MARK] When does the segmentation fault message appear? Check the appropriate answer or answers.

- ☐ Compile time
- ☑ Run time
- ☐ When runnning `make print_ftree`

## Question 3. [4 MARKS]

Consider the following Makefile from Assignment 1

```
FLAGS = -Wall -std=c99

compute_hash: compute_hash.o hash_functions.o
        gcc ${FLAGS} -o $@ $^

%.o : %.c
        gcc ${FLAGS} -c $<

clean :
        rm *.o compute_hash
```

The contents of the current working directory are:

```
Makefile    compute_hash.c  hash_functions.c
```

In the questions below, identify the files that are created, modified or deleted when each command is run in sequence.

### Part (a) [1 MARK] `make compute_hash.o`

| Created | Modified | Deleted |
|---|---|---|
| `compute_hash.o` | | |

### Part (b) [1 MARK] `make`

| Created | Modified | Deleted |
|---|---|---|
| `hash_functions.o`<br>`compute_hash` | | |

### Part (c) [1 MARK]

First `hash_functions.c` is modified and then `make compute_hash` is run.

| Created | Modified | Deleted |
|---|---|---|
| | `(hash_functions.c)`<br>`hash_functions.o`<br>`compute_hash` | |

### Part (d) [1 MARK]

I do some more work on the program and add a new file called `helper.c` that contains functions called by code in `compute_hash.c`. (Assume I updated the source files correctly.) Modify the Makefile above so that the new file will be compiled correctly with the program.

*Add helper.o to the list of prerequistes for the compute_hash rule*

## Question 4. [6 MARKS]

```
struct team {
    char name[16];
    char *players[4];
};
```

```
struct team *ont;
struct team scotties[10];
```

### Part (a) [3 MARKS]

Give the declaration for the variable x in each of the statements below, or if there is an error in the statments, describe the error in a few words. Assume that `ont` and `scotties` (above) have been appropriately initialized with all necessary memory allocated.

| Statement | Declaration of x or error |
|---|---|
| `x = ont->name;` | `char *x` |
| `x = ont->name[0];` | `char x` |
| `x = &ont;` | `struct team **x` |
| `x = scotties[2];` | `struct team x` |
| `x = scotties.players[1];` | scotties isn't a struct |
| `x = &scotties[2].players[1];` | `char **x` |

### Part (b) [3 MARKS]

For each of the following independent snippet of code identify the problem with the code, or write "no problem" if there are no potential problems with the snippet.

| Statements | Problem |
|---|---|
| `ont->name = malloc(10);`<br>`strncpy(ont->name, "Homan", strlen("Homan"));` | Can't reassign an array<br>or Result is not terminated |
| `ont->name = malloc(10);`<br>`ont->name = "Homan";` | Can't reassign an array<br>Memory leak |
| `char name[10] = "Homan";`<br>`ont->name = name;` | Can't reassign an array<br>Probably don't want to set the field to stack memory. |

## Question 5. [6 MARKS]

Complete the function below, so that the following examples will work correctly and the minimum amount of memory is used. The only string library functions you may use are `strlen` and `strncpy`.

```
char str[] = "first,,third,fourth";
printf("0 %s\n", getfield(str, 0));
printf("1 %s\n", getfield(str, 1));
printf("2 %s\n", getfield(str, 2));
printf("3 %s\n", getfield(str, 3));
```

Prints:

```
0 first,,third,fourth
1 ,third,fourth
2 third,fourth
3 fourth
```

```
/* Returns a new string containing the characters starting from
 * the nth comma in line until the end of the line.
 * Does not modify line.
 * Returns NULL if n is larger than the number of commas in line
 */

char *getfield(char *line, int n) {

    char *cut_nth(char *line, int n) {
        char *result = malloc(strlen(line)+1);

        /* find the starting point */
        int i = 0;
        while(line[i] != '\0' && n > 0) {
            if(line[i] == ',')
                n--;
            i++;
        }

        if(line[i] == '\0') {
            return NULL;
        }

        int j = 0;
        while(line[i] != '\0') { // Could use strncpy here
            result[j] = line[i];
            j++; i++;
        }
        result[j] = '\0';

        return result;
    }
```