Question 1. [10 marks]

Part (a) [8 marks]

Complete the following program according to the instructions in the comments. Assume that all system calls succeed and that the arguments are passed correctly.

```
struct graph {
    char *title;
    int size;
    int *data;
};

int main(int argc, char **argv) {
    if(argc != 4){
        fprintf(stderr, "Usage: mkgraph TITLE NUM_POINTS FILE\n");
    }

    // Allocate space on the heap for a struct graph and save the pointer to
    // this memory in a variable called graph
    struct graph *graph;
    graph = malloc(sizeof(struct graph));

    // Set the graph's size field to the integer value of the second command
    // line argument
    graph->size = strtol(argv[2], NULL, 10);

    // Set graph's title to the first command line argument without copying it
    graph->title = argv[1];

    // Initialize the graph's data field to refer to heap-allocated space for
    // size number of elements.
    graph->data = malloc(graph->size * sizeof(int));

    // Open the file given by file name in the third command line argument
    // The file will be opened for reading.
    FILE *fp = fopen(argv[3], "r");

    // The open file is a text file containing  numbers separated by spaces
    // Read each number from the file and store it in the data field of graph
    // in the order they are read
    int sum = 0;
    for(int i = 0; i < graph->size; i++) {
        fscanf(fp, "%d", &graph->data[i]);
        sum += graph->data[i];
    }

    // Compute the average of the elements and print it in the statement below
    printf("%f\n", (float)sum/graph->size);
    return 0;
}
```

Part (b) [2 marks]

Check the box beside each of the free statements below that correctly free memory used in the above program. For each line that is not checked, briefly explan why the statement is incorrect.

☑ `free(graph);`

☐ `free(graph->title);`
Set to argv[1] which is not necessarily heap allocated

☑ `free(graph->data);`

☐
```
for(int i = 0; i < graph->size; i++) {
    free(graph->data[i]);
}
```
Each element was not malloc'd separately, so we can't free them separately.

Question 2. [1 mark]

Write a line to execute `compute_hash` with a blocksize argument of 4 and redirect standard input so that the program reads from the file `test.in` rather than from standard input.

```
compute_hash 4 < test.in
OR
./compute_hash 4 < test.in
```

Question 3. [3 marks]

Your current working directory has two files prog.c and helper.c. prog.c has a main function in it and calls functions from helper.c

Write a Makefile with one rule so that when you type `make prog`, the executable prog will be created only if one or both of the files prog.c and helper.c have been modified. Note that the program should be compiled with the -Wall and -g flags.

```
prog : prog.c helper.c
    gcc -Wall -g -o prog prog.c helper.c
```

Question 4. [2 marks]

I spent a long time working on my `print_ftree` program on cdf and it works perfectly, but when I try it on my Linux machine at home, once it 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 5. [4 marks]

For each of the code fragments below, there is missing code. At the very least, the line (or lines) that declare and possibly initialize the variable `x` are missing. If the code will not compile no matter what you put for the missing code, check `COMPILE ERROR` and explain why. If the code will compile, but is not guaranteed to run without an error, check `RUN-TIME ERROR` and explain why. Otherwise, check `NO ERROR` and give the correct declaration for x. You don't need to show any other missing code. The first one is done for you.

Code Fragment	ERROR	Declaration for `x` or explanation

`int y;`
`// missing code`
`x = y;`
	[✓ NO ERROR]
	☐ COMPILE ERROR
	☐ RUN-TIME ERROR
		`int x;`

`char input[6];`
`// missing code`
`strcpy(input, x);`
	[✓ NO ERROR]
	☐ COMPILE ERROR
	☐ RUN-TIME ERROR
		It depends.
		Could be no error

`char *film = "Moonlight";`
`// missing code`
`x = film;`
`x[0] = 'L';`
	☐ NO ERROR
	☐ COMPILE ERROR
	[✓ RUN-TIME ERROR]
		Atempting to change a literal

`char *args[3];`
`// missing code`
`x = *args[2];`
	[✓ NO ERROR]
	☐ COMPILE ERROR
	☐ RUN-TIME ERROR
		char x;

`// struct location is defined`
` struct location src;`
`// missing code`
`src = x;`
	[✓ NO ERROR]
	☐ COMPILE ERROR
	☐ RUN-TIME ERROR
		struct location x

`int *mkpoint(int x, int y) {`
`    int pt[2] = {x, y};`
`    return pt;`
`}`
`x = mkpoint(3, 4);`
	☐ NO ERROR
	☐ COMPILE ERROR
	[✓ RUN-TIME ERROR]
		int *x
		returning stack memory

Question 6. [6 marks]

Complete the function below, so that the following examples will work correctly and the minimum amount of memory is used. You may only use the string library functions `strlen` and `strncpy`. (You are not required to use these functions.)

```
char *path1 = "/usr/include";
char *result1 = basename(path1);

char *path2 = "/usr/include/";
char *result2 = basename(path2);

char *path3 = "file.txt";
char *result3 = basename(path3);

char *path4 = "./a2/test/test1.in";
char *result4 = basename(path4);
printf("%s, %s, %s, %s\n",
    result1, result2, result3, result4);
```

Prints:

```
include, , file.txt, test1.in
```

```
char *basename(char *path) {
    int last = 0;
    int i;
    // find the last slash
    for(i = 0; path[i] != '\0'; i++) {
        if(path[i] == '/') {
            last = i+1;
        }
    }

    char *result = malloc(strlen(&path[last]) + 1);
    strncpy(result, &path[last], strlen(&path[last]) + 1);
    return result;
}
```