## Question 1. [5 MARKS]

Assume you have a terminal open, and the current working directory contains a C program file called `wizard.c` and a text file called `potion.txt`.

### Part (a) [1 MARK]

The following command is used to compile `wizard.c` into an executable:
`gcc -Wall -o magic -g -std=gnu99 wizard.c`

Write a shell command to execute the executable file produced by the statement above with input redirected so the program reads from `potion.txt` rather than from standard input.

```
./magic < potion.txt
OR
magic < potion.txt
```

### Part (b) [2 MARKS]

Now, a header file named `wizard.h` is added to the current working directory and `wizard.c` is revised to include that header file.

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

```
wizard: wizard.c wizard.h
    gcc -Wall -o wizard -g wizard.c
OR
wizard: wizard.c wizard.h
    gcc -Wall -o magic -g wizard.c

(including std=gnu99 was ok but not required)
```

### Part (c) [1 MARK]

The following command is used to set the permissions of `potion.txt`:
`chmod 731 potion.txt`

Check the boxes to indicate the file permissions of `potion.txt` after the command above has been executed.

| | | | |
|---|---|---|---|
| user: | ☒ read | ☒ write | ☒ execute |
| group: | ☐ read | ☒ write | ☒ execute |
| other: | ☐ read | ☐ write | ☒ execute |

### Part (d) [1 MARK]

In the box, print the number of bytes that will be written to the file by this code fragment.

```
int i = 82;
fprintf(fp, "%d\n", i);
```

3 OR sizeof(char) * 3

## Question 2. [6 MARKS]

For each code fragment below, if the code will not compile or will generate a warning when compiled with the `-Wall` flag, 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 show what is printed. The first one is done for you.

| Code Fragment | ERROR | Output or explanation for error |
|---|---|---|
| `int y = 2;`<br>`int x = y;`<br>`printf("%d %d", x, y);` | ☑NO ERROR<br>☐ COMPILE ERROR<br>☐ RUN-TIME ERROR | `2 2` |
| `char *name = "Me";`<br>`printf("%s\n", name);`<br>`free(name);` | ☐ NO ERROR<br>☐ COMPILE ERROR<br>☑RUN-TIME ERROR | name is on the stack and cannot be freed |
| `char *animal = "cat";`<br>`animal[0] = 'r';`<br>`printf("%s", animal);` | ☐ NO ERROR<br>☐ COMPILE ERROR<br>☑RUN-TIME ERROR | cannot mutate a string literal |
| `char s[] = "RaCecAr";`<br>`char *p = s;`<br>`int q = 6;`<br>`while (p <= &s[q]) {`<br>`  if (*p != s[q]) {`<br>`  *p = s[q];`<br>`  }`<br>`  p++;`<br>`  q--;`<br>`}`<br>`printf("%s", s);` | ☑NO ERROR<br>☐ COMPILE ERROR<br>☐ RUN-TIME ERROR | rAcecAr |
| `char *w = malloc(3 * sizeof(char));`<br>`strcpy(w, "AB");`<br>`char *m[2];`<br>`m[0] = w;`<br>`m + 1 = w + 1;`<br>`printf("%s, %s", *m, *(m + 1));` | ☐ NO ERROR<br>☑COMPILE ERROR<br>☐ RUN-TIME ERROR | `"m + 1" should be "*(m + 1)"` |
| `char food[6];`<br>`char *fruit = "banana";`<br>`food = fruit;`<br>`printf("%s", food);` | ☐ NO ERROR<br>☑COMPILE ERROR<br>☐ RUN-TIME ERROR | `incompatible types (char[6] vs. char *)` |
| `char s[10] = "computer";`<br>`s[6] = '\0';`<br>`printf("%s", s);` | ☐ NO ERROR<br>☑COMPILE ERROR<br>☐ RUN-TIME ERROR | `comput` |

## Question 3. [5 MARKS]

Write the function `count_occurrences` according to its description below. Complete the main function.

Notice that it counts overlapping occurences. For example, given `"ababa"` and `"aba"` for the first two arguments, the function should update the memory pointed at by `result` to `2`.

You may not use `strstr()`, nor any other string library function that would find a substring for you.

```
/* Find the number of times substr appears in str (including overlapping occurrences) and
 * set the memory pointed at by result to that number.  If str and substr are identical,
 * set the memory pointed at by result to 1. */
void count_occurrences(char *str, char *substr, int *result) {

    int last = strlen(str) - strlen(substr) + 1;
    int count = 0;
    int flag;

    for (int i = 0; i < last; i++) {
        flag = 1;

        for (int j = i; j < (i + strlen(substr)); j++) {
            if (str[j] != substr[j-i]) {
                flag = 0;
                break;
            }
        }
        if (flag == 1) {
            count++;
        }
    }

    *result = count;
}

int main() {

    int res;
    // Call count_occurrences on "ababa" to find "aba" and set res.
    count_occurrences("ababa", "aba", &res);


    printf("aba occurred %d times in ababa\n", res);

    return 0;
}
```

## Question 4. [9 MARKS]

Complete the following program according to the instructions in the comments. Assume that all system calls succeed and that the arguments are of the specified format. Only allocate the space you need.

```
struct flight {
    char *code;
    int seats_available;
};
/* Return a pointer to a new struct flight with its code and seats_available
 * initialized to the data given in arg.  arg will be a flight code (e.g., AC1123)
 * followed by a colon (:) and the number of seats on that flight (e.g., 100).
 * Once a flight is created, its code must refer to a dynamically-allocated string. */
struct flight *create_flight(char *arg) {

    struct flight *new_flight = malloc(sizeof(struct flight));

    // argument format: AC1123:24
    char *tmp = strchr(arg, ':');
    int seats = strtol(tmp + 1, NULL, 10);
    int size = strlen(arg) - strlen(tmp) + 1;

    new_flight->code = malloc(sizeof(char) * size);
    strncpy(new_flight->code, arg, size - 1);
    new_flight->code[size] = '\0';

    new_flight->seats_available = seats;

    return new_flight;

}
/* If there are enough seats available on f, book num_seats; Otherwise do nothing. */
void book_seat(struct flight* f, int num_seats) {

    if (f->seats_available >= num_seats) {
        f->seats_available -= num_seats;
    }

}
int main(int argc, char **argv) {
    // argv[1] represents a flight in the format code:seats (e.g., AC1123:100, WSG324:24)
    struct flight *my_flight = create_flight(argv[1]);
    book_seat(my_flight, 2);
    // Free memory

    free(my_flight->code);
    free(my_flight);
}
```