Question 1. [4 MARKS]

Assume you have a terminal open, and the current working directory contains a C program file called `args.c`. The contents of the file are shown below:

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("%d\n", argc);
    return 0;
}
```

Part (a) [1 MARK] Write a command to compile `args.c` into an executable called `args`, using the `gnu99` standard and including the flag to display all warning messages.

```
gcc -Wall -std=gnu99 -o args args.c
```

Part (b) [1 MARK] Write the output of the program for each of the following invocations:

```
./args
```

```
1
```

```
./args abc 123 xyz
```

```
4
```

Part (c) [1 MARK]

Write a command that invokes `args` redirecting the program's standard output to a file called `data`.

```
./args > data  OR args > data
```

Part (d) [1 MARK]

Write a single unix command to set the permissions of the file `args` to `rwxr-xrw-`.

```
chmod 756 args  OR chmod u=rwx,g=rx,o=rw args
```

Question 2. [3 MARKS]

Suppose that the current working directory contains only the files:

- `helpers.c`: contains helper functions
- `helpers.h`: contains the prototypes for those helper functions, and
- `life.c`: contains a `main` function that calls on the helper functions.

Write the commands needed to compile the code to produce object files `helpers.o` and `life.o`, and then use the object files to produce an executable named `mylife`.

(The first two commands can be reversed.)

```
gcc -c helpers.c
gcc -c life.c
gcc -o mylife helpers.o life.o
```

Question 3. [3 MARKS]

The following program runs without errors. Print its output neatly in the box provided.

% A pointer to a local variable does not affect the original.

```
int func(int a, int *b) {
    int *ptr = &a;
    *ptr += 5;

    ptr = b;
    *ptr -= 3;

    return a;
}

int main() {
    int x = 2;
    int y = 8;
    int ret = func(x, &y);

    printf("x: %d\n", x);
    printf("y: %d\n", y);
    printf("ret: %d\n", ret);

    return 0;
}
```

Answer:

```
x: 2
y: 5
ret: 7
```

Question 4. [8 marks]

Consider the code and memory diagram below.

Part (a) [6 marks]

Fill in the memory diagram to show the current state of the program exactly before the return statement on **line 12** is executed. If there are uninitialized blocks of memory at that point in the program, write their values as ???.

```
1   char **split(char *s) {
2       char *ptr = strchr(s, '.') + 1;
3   
4       char **tokens = malloc(2 * sizeof(char *));
5       tokens[0] = malloc(ptr - s);
6       strncpy(tokens[0], s, ptr-s-1);
7       tokens[0][ptr-s-1] = '\0';
8   
9       tokens[1] = malloc(strlen(ptr) + 1);
10      strcpy(tokens[1], ptr);
11  
12      return tokens;
13  }
14  
15  int main(void) {
16      char **arr = split("out.txt");
17      printf("%s\n", arr[1]);
18  
19      // TODO: Free the allocated memory.
20  
21      return 0;
22  }
```

Section	Address	Value	Label
Read-only	0x100	out.	
	0x104	txt\0	
	0x108		
	0x10c		
	0x110		
	0x114		
	0x118		
	0x11c		
	⋮	⋮	
Heap	0x23c	0x24c	
	0x240		
	0x244	0x250	
	0x248		
	0x24c	out\0	
	0x250	txt\0	
	0x254		
	0x258		
	0x25c		
	⋮	⋮	
split	0x454	0x100	s
	0x458		
	0x45c	0x104	ptr
	0x460		
	0x464	0x23c	tokens
	0x468		
main	0x46c	???	arr
	0x470		
	0x474		
	0x478		
	0x47c		

Part (b) [2 marks]

Add the necessary statement(s) that would follow line 19 to properly free the memory allocated by the program:

```
// The first two statements can be reversed.
free(arr[0]);
free(arr[1]);
free(arr);
```

Question 5. [2 marks]

The following code snippet runs without errors. Print its output neatly in the box provided.

```
struct Car {
    char *color;
    int mileage;
};

void update_mileage(struct Car c, struct Car *c_ptr) {
    c_ptr->mileage += 500;
    c.mileage += 200;
}


int main() {
    struct Car car;
    char *color_ptr = "Green";
    car.color = color_ptr;
    car.mileage = 1000;

    color_ptr = "Blue";
    struct Car *car_ptr = &car;
    car_ptr->mileage = 1500;

    printf("(%s, %d)\n", car.color, car.mileage);

    update_mileage(car, &car);

    printf("(%s, %d)\n", car_ptr->color, car_ptr->mileage);

    return 0;
}
```

```
(Green, 1500)
(Green, 2000)
```

Question 6. [5 marks]

The question is based on the following linked list definition:

```
struct node {
    int ID;
    char *name; // Points to a dynamically allocated string.
    struct node *next;
};
```

Considering that the name of each linked list node has the form `"lastname, firstname"`, for each node starting at the specified `head`, reorder the two names and convert them into the following form: `"firstname-lastname"`. Write your code so that it does not have a memory leak.

```
void format_name(struct node *head) {
    char *ptr, *str;

    while (head) {
        ptr = strchr(head->name, ',');

        str = malloc(strlen(head->name)); // Also full marks: strlen(head->name) + 1
        strcpy(str, ptr + 2);
        strcat(str, "-");
        strncat(str, head->name, ptr - head->name);

        free(head->name);
        head->name = str;

        head = head->next;
    }
}
```