Question 1. [3 MARKS]

Assume you have a terminal open, and the current working directory contains a C file called `nums.c`.

Part (a) [1 MARK]

Write a command to compile `nums.c` into an executable called `nums`, including debug symbols and the flag to display all warning messages.

```
gcc -Wall -g -o nums nums.c
```

Part (b) [1 MARK]

Write a command that invokes `nums` and pipes the output to the program `sort`.

```
./nums | sort
```

Part (c) [1 MARK]

The following command is used to set the permissions of `file.txt`:

```
$ chmod 643 file.txt
```

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

user:	☒ read	☒ write	□ execute
group:	☒ read	□ write	□ execute
other:	□ read	☒ write	☒ execute

Question 2. [3 marks]

Consider the following makefile:

```
FLAGS = -Wall -g -std=gnu99
DEPENDENCIES = queue.h

all: help_centre print_queue

help_centre: help_centre.o queue.o
        gcc ${FLAGS} -o $@ $^

print_queue: print_queue.o queue.o
        gcc ${FLAGS} -o $@ $^

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

Suppose that the only files in the current working directory are the source files, the header file, and the Makefile. Write the actions that are executed (in the order that they are executed), when we run:
`make help_centre`

(The first two commands can be reversed.)

```
gcc -Wall -g -std=gnu99 -c help_centre.c
gcc -Wall -g -std=gnu99 -c queue.c
gcc -Wall -g -std=gnu99 -o help_centre help_centre.o queue.o
```

Makefile variables:

`$@` target
`$^` all prerequisites
`$?` all out of date prerequisites
`$<` first prerequisite

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.

```
void func(int a, int *b, int *c) {
    int d = a;
    int *ptr = &a;
    *ptr -= 2;

    b = malloc(sizeof(int));
    *b = a;
    *c = d;
}

int main() {
    int x = 6;
    int *y = NULL;
    int ret;
    func(x, y, &ret);

    printf("x: %d\n", x);
    if (y == NULL)
        printf("y is NULL\n");
    else {
        printf("y: %d\n", *y);
        free(y);
    }
    printf("ret: %d\n", ret);

    return 0;
}
```

Answer:

```
x: 6
y is NULL
ret: 6
```

Question 4. [9 MARKS]

Consider the code and memory diagram below.

Part (a) [8 MARKS]

Fill in the memory diagram to show the current state of the program exactly before the return statement on **line 7** is executed. If there are uninitialized blocks of memory at that point in the program, write their values as ???.

```
1   char *truncate(char *s, int new_size) {
2       if (new_size >= strlen(s))
3           return 0;
4
5       s[new_size] = '\0';
6
7       return &s[new_size + 1];
8   }
9
10  int main(void) {
11      int len = strlen("CSC209H1-S1");
12      char *str = malloc(len + 1);
13      strcpy(str, "CSC209H1-S1");
14
15      char *ptr = strchr(str, '-');
16      char *chars = truncate(str, ptr - str);
17      printf("%s, %s\n", str, chars);
18      free(str);
19      return 0;
20  }
21
```

Section	Address	Value	Label
Read-only	0x100	CSC2	
	0x104	09H1	
	0x108	-S1\0	
	0x10c		
	0x110		
	0x114		
	0x118		
	0x11c		
	⋮	⋮	
Heap	0x23c	CSC2	
	0x240	09H1	
	0x244	\0S1\0	
	0x248		
	0x24c		
	0x250		
	0x254		
	0x258		
	⋮	⋮	
truncate	0x448		
	0x44c		
	0x450		
	0x454	8	new_size
	0x458	0x23c	s
	0x45c		
main	0x460	???	chars
	0x464		
	0x468	0x244	ptr
	0x46c		
	0x470	0x23c	str
	0x474		
	0x478	11	len

Part (b) [1 MARK]

How many bytes are freed by the free statement on line 18? ______________________

12

Question 5. [2 marks]

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

```
struct Airplane {
    char *destination;
    int passengers;
};

void update_information(struct Airplane *a_ptr) {
    a_ptr->destination[2] = 'C';
    a_ptr->passengers += 50;
}

int main(void) {
    struct Airplane airplane;
    airplane.destination = malloc(strlen("YYZ") + 1);
    strcpy(airplane.destination, "YYZ");
    airplane.passengers = 100;

    struct Airplane *airplane2_ptr = malloc(sizeof(struct Airplane));
    airplane2_ptr->destination = airplane.destination;
    airplane2_ptr->passengers = airplane.passengers;
    update_information(airplane2_ptr);

    printf("(%s, %d)\n", airplane.destination, airplane.passengers);
    printf("(%s, %d)\n", airplane2_ptr->destination, airplane2_ptr->passengers);

    return 0;
}
```

```
(YYC, 100)
(YYC, 150)
```

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;
};
```

Implement a function that iterates over the nodes of a linked list starting at the specified `head` and modifies the `name` of every node in the list by adding `n` additional copies of the name. For example, if a name is originally *Marcia* and `n` is 2, the name becomes *MarciaMarciaMarcia*.

Write your code so that it does not have a memory leak.

```
void repeat_name(struct node *head, int n) {
    char *ptr;
    int i, len;

    while (head) {
        len = strlen(head->name);
        ptr = head->name;

        head->name = malloc((n + 1) * len + 1);
        head->name[0] = '\0';
        for (i = 0; i <= n; i++)
            strcat(head->name, ptr);

        free(ptr);
        head = head->next;
    }
}
```