Question 1. [4 MARKS]

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

Part (a) [1 MARK]

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

```
gcc -Wall -o count count.c
```

Part (b) [1 MARK]

Write a command that invokes `count`, redirecting its output to a file called `result.txt`

```
./count > result.txt    OR     count > result.txt
```

Part (c) [1 MARK]

Assume that the absolute path to the current working directory is `/h/user/d1`. Invoke the `count` program providing a relative path to `/h/user/d2/f1` as a command-line argument.

```
./count ../d2/f1
```

Part (d) [1 MARK]

Write a shell command to set the permissions of `my_file` so that it is writable by its owner, readable and executable by anyone in the file's group and executable by other. Don't assume anything about its permissions before your command is run.

```
chmod u=w,g=rx,o=x my_file
OR
chmod 251 my_file
```

Question 2. [3 MARKS]

Consider the following makefile:

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

all: path_finder display_graph

path_finder: path_finder.o graph.o
        gcc ${FLAGS} -o $@ $^

display_graph: display_graph.o graph.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 display_graph`

(The first two commands can be reversed.)

```
gcc -Wall -g -std=gnu99 -c display_graph.c
gcc -Wall -g -std=gnu99 -c graph.c
gcc -Wall -g -std=gnu99 -o display_graph display_graph.o graph.o
```

Makefile variables:

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`$?` all out of date prerequisites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

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 one.

```
#include <stdio.h>

void func(int a, int *b, int **c) {
    int *ptr = &a;
    a++;

    *c = b;
    ptr = b;

    *ptr = a - 2;
}

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

    func(x, &ret, &y);

    printf("x: %d\n", x);
    printf("y: %d\n", *y);
    printf("ret: %d\n", ret);
    return 0;
}
```

Answer:

```
x: 6
y: 5
ret: 5
```

Question 4. [8 MARKS]

Consider the code and memory diagram below.

Part (a) [7 MARKS]

Fill in the memory diagram to show the current state of the program exactly before the return statement on **line 9** is executed. If there are uninitialized blocks of memory at that point in the program, write their values as `???`.

```
1  char **divide(char *s) {
2      char *ptr = strchr(s, 'a');
3      *ptr = '\0';
4
5      char **tokens = malloc(2 * sizeof(char *));
6      tokens[0] = s;
7      tokens[1] = ptr + 1;
8
9      return tokens;
10 }
11
12 int main(void) {
13     char str[] = "overall";
14     char **arr = divide(str);
15     printf("%s%s\n", arr[0], arr[1]);
16     free(arr);
17     return 0;
18 }
```

Section	Address	Value	Label
Read-only	0x100	over	
	0x104	all\0	
	0x108		
	0x10c		
	0x110		
	0x114		
	0x118		
	0x11c		
	⋮	⋮	
Heap	0x23c	0x480	
	0x240		
	0x244	0x485	
	0x248		
	0x24c		
	0x250		
	0x254		
	0x258		
	0x25c		
	⋮	⋮	
	0x454		
	0x458		
	0x45c		
divide	0x460	0x23c	`tokens`
	0x464		
	0x468	0x484	`ptr`
	0x46c		
	0x470	0x480	`s`
	0x474		
main	0x478	???	`arr`
	0x47c		
	0x480	over	`str`
	0x484	all\0	

Part (b) [1 MARK]

How many bytes are freed by the `free` statement on line 16? _____

16

Question 5. [2 MARKS]

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

```
#define MAX_NAME 16

struct Car {
    char color[MAX_NAME + 1];
    int mileage;
};

void update_information(struct Car *c_ptr) {
    c_ptr->color[0] = 'B';
    c_ptr->mileage += 50;
}

int main(void) {
    struct Car car;
    strcpy(car.color, "Green");
    car.mileage = 5000;

    struct Car *car_ptr = malloc(sizeof(struct Car));
    memcpy(car_ptr, &car, sizeof(struct Car));
    update_information(car_ptr);

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

    return 0;
}
```

(Green, 5000)
(Breen, 5050)

Question 6. [5 MARKS]

This 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 searches over a linked list for the specified node `ID`. In case the `ID` exists, then the function replaces the node's current `name` with a dynamically-allocated string containing the current `name` followed by a *space* and the `nickname` argument. The function returns the old name. If the `ID` is not found or the list is empty, the function returns *NULL*. The memory allocated for the new string should be exactly the right size to fit the new string.

% Linked list traversal.
% String parameter (involves *malloc*).
% Change the fields of a struct.

```
char *add_nickname(struct node *head, int nodeID, char *nickname) {
    char *oldname = NULL;

    while (head) {
        if (head->ID == nodeID) {
            oldname = head->name;
            head->name = malloc(strlen(oldname) + strlen(nickname) + 2);
            strcpy(head->name, oldname);
            strcat(head->name, " ");
            strcat(head->name, nickname);
            break;
        }
        head = head->next;
    }

    return oldname;
}
```