Question 1:

a) ascii

```
#include <stdio.h>
int main(void){
        FILE * fp = fopen("names.txt", "r");
        int i;
        for(i = 0; i < 100; i++){
                char name[12];
                fgets(name, 12, fp);
                puts(name);
        }
}
```

b) array

```
#include <stdio.h>
#include <string.h>
int main(void){
        FILE * fp = fopen("names.txt", "r");
        char * names[100];
        int max = 0;
        int i;
        for(i = 0; i < 100; i++){
                char name[12];
                fgets(name, 12, fp);
                puts(name);
                names[i] = name;
                if (strlen(name) > max){
                        max = strlen(name);
                }
        }
}
```

c) Linux compilation

```
gcc -lm -o simulation source1.c source2.c source3.c source4.c
```

✓ ✓ ✓ ✓ 4

d) Makefile

```
CC = gcc
simulation: source1.o source2.o source3.o source4.o
	echo “Creating simulation executable from all object files”
	$(CC) $^ -o $@
%.o: %.c
	echo “Creating object files from source files”
	$(CC) -lm -c $<
```

✓ 0.5 ✓ 0.5 ✓ ✓ 0.5 ✓ ✓ ✓ 0.5 ✓ 6

10/10

Question 2

a) Command line arguments

```
#!/bin/bash
echo “Number of args: $#”
for arg in $@
do
	echo $arg
done
exit 0
```

✓ ✓ ✓ ✓ 4

b) executing files

```
#!/bin/bash

files=$(ls)
for file in $files
do
	# Check if the file is a directory
```

✓ ✓

```
        if [ -d $file ]; then
                cd $file
                echo “cd’ed into $(file)”
        # Check if it's a regular file
        elif [ -f $file ]; then
                if [ -x $file ]; then
                        ./$(file)
                else
                        exit 1
                fi
        fi
done
exit 0
```

8

1

c) grep prints lines that match a given pattern using regular expressions. ls | grep “\.c” grep "#/bin/bash" *

d) “%” matches anything but NULL and remembers what it matched. “^” is the list of prerequisites for the target being compiled. 1

e) -g compiles with the debugging option. -Wall shows all compilation warnings. -O2 is the recommended amount of optimisation – not too long, but optimises quite a bit. 3

16/20

Question 3

a)

```
char line[100];
fgets(line, sizeof(line), stdin);
sscanf(line, “%d %d”, &list[i].re, &list[i].im);
```

b) == compares the equality of the values, && is the and logic operator, || is or, and ++ increments by 1.

c) Add “b” in the fopen() mode, add #include <string.h> at the top, change fprintf to

```
fwrite(sum.re, sizeof(int), 1, out_file_ptr);
fwrite(sum.im, sizeof(int), 1, out_file_ptr);
```

d) // for one-line comments, between /* and */ for multi-line comments.

```
/* Calculates the sum of 5 complex numbers that the user inputs and outputs the sum to output.dat
 * Uses a complex number struct, of which the main one is “sum”, out_file_ptr is the pointer to the
 * output file, “list” is an array of structs.
 * Limitations: uses set number of input (5) and reuses it as an integer literal (bad for maintainability)
 */
```

e)

```
struct complex sum = {.re=0, .im=0};
```

f)

As it is declared in file scope, the kernel allocates a memory location reserved for the list, where the list will be contained. It will also be visible and accessible for other functions in the file.