Question 1 (11 marks)

For each of the following type declarations:

- Briefly explain how the newly-defined datatype is organised in memory.
- Give an example of **declaring and initialising a single variable** of that type.

(a) [2 marks]

```
union Travel {
    float time;
    int space[2];
};
```

(b) [2 marks]

```
typedef enum { WIBBLY, WOBBLY = 20,
               TIMEY = 25, WIMEY } Stuff;
```

(c) [3 marks]

```
struct Poosh {
    void (*medusa)();
    const int* cascade;
};
```

(d) [4 marks]

```
typedef struct River {
    char* forest;
    struct River* water;
} Pond;
```

Question 2 (9 marks)

Write a short (but complete) C program to output a random integer within a range given on the command-line. Your program should accept two command-line parameters, representing the lower and upper bounds of the number to print.

For instance, your program may be called as follows:

```
./printrandom 5 15
```

Given this command-line, your program should print a random integer between 5 and 15, inclusive. Your program should (in general) output *different* integers when called the same way multiple times.

You *do not* need to perform error checking on the parameters. You may assume the program is always invoked with two valid parameter values.

Question 3 appears on the next page

Question 3 (20 marks)

Consider the following code.

```
int a[6] = {-2, -1, 0, 1, 2, 3};
int b[5] = {0, 5, 10, 15, 20};

int** s = NULL;
int* t = NULL;
int** y = NULL;

s = (int**)malloc(2 * sizeof(int*));
t = &a[a[b[1]]];
y = (int**)malloc(sizeof(int*));

s[0] = b + 1;
*(s + 1) = *s + 1;
*y = s[1] + 1;

t[1] = s[0][1] + s[1][1];
(*(*y + 1))++;
```

Based on this:

(a) Draw a diagram showing all the pointer relationships created. [16 marks]

(b) Show the contents of `a` and `b` at the end. [4 marks]

Question 4 (20 marks)

The following function (on the next page) implements a slideshow viewer. The function asks the user for a list of image files to display. It displays them sequentially, showing each for 10 seconds before hiding it and moving on to the next. If the user selects any invalid image files, the function skips over these without delay, printing out an error message.

However, the program has defects! The defects are **not in the code shown here**, but rather in the functions *called* by `slideshow()`.

Question 4 continues on the next page

```
1  #define DELAY_LENGTH 10
2
3  void slideshow() {
4      char** files;
5      int length, image, handle;
6
7      /* Ask the user to select several image files from a menu */
8      files = selectImages(&length);
9
10     for(image = 0; image < length; image++)
11     {
12         handle = showImage(files[image]);
13         if(handle != 0) {
14             /* If the image can be successfully displayed,
15                wait for a while, then hide it again. */
16             delay(DELAY_LENGTH);
17             hideImage(files[image]);
18         }
19         else {
20             printf("%s could not be displayed\n", files[image]);
21         }
22     }
23
24     freeStrings(files, length);
25 }
```

For each situation below:

- State your initial hypothesis, and justify it. What possible fault/defect (in the code *not* shown) might explain the observations?
- Explain how you would use debugger features to confirm or refute your hypothesis. In particular, where would you set **breakpoints**, and which **variables** would you monitor?

Your hypothesis should be plausible, and your explanation should follow logically. State any relevant (and realistic) assumptions you make about the code not shown.

(a) The function appears to work, except that the last image selected is never displayed. [5 marks]

(b) A segmentation fault occurs after the user selects the images. No images are displayed. [5 marks]

(c) When the user selects a set of valid images, the program displays all of them without delay, and also reports that each one could *not* be displayed. [5 marks]

(d) The first image is shown for ten seconds, and then a segmentation fault occurs. [5 marks]

Question 5 appears on the next page

Question 5 (40 marks)

For this question, ensure all your code conforms to the characteristics emphasised in the lectures and tutorials.

(a) Declare suitable C datatypes to represent each of the following sets of information (as you would do in a header file):

(i) A list of privacy settings for a social media application:

- Whether or not to display your email address.
- Whether or not to display your messages publicly.

(ii) A social media profile for a person, consisting of:

- The person's name.
- The person's email address.
- The number of other profiles/people this person "follows".
- An array containing pointers to these other profiles.
- A set of privacy settings for this profile.

You may assume that names and email addresses are no more than 100 characters long. However, there is no fixed limit to the number of people someone may follow.

[8 marks]

(b) Write a C function called `readProfiles` to read text files in the following format:

- The first line of the file gives the total number of profiles (a single integer).
- Each subsequent line contains the following information, separated by spaces:
  - a name (without spaces);
  - an email address (without spaces);
  - n — the number of people followed (an integer);
  - the email privacy setting (either '`Y`' or '`N`'); and
  - the message privacy setting (either '`Y`' or '`N`').

For example:

```
4
Fred freddy@gmail.com 2 Y N
Joe  jkl@hotmail.com  1 N N
Sam  sam@sam.org      2 Y Y
Les  les@isp.net      3 N N
```

Each person follows the **first** n people in the list, not counting themselves. In the example, Fred follows 2 people, Joe 1 person, Sam 2 people, and Les 3 people. Thus, Fred follows Joe and Sam, Joe follows Fred, Sam follows Fred and Joe, and Les follows Fred, Joe and Sam.

Question 5 part b continues on the next page

Your `readProfiles` function should do the following:

- Import an input filename (referring to a file with the above format) as a `char` pointer.
- Open the file for reading.
- Dynamically allocate the appropriate memory for an array of the required data structures from part (a).
- Read the data from the file into the array, filling in the "follows" links as appropriate.
- Export the length of the array via a pointer parameter.
- Return a pointer to the array.
- Return `NULL` instead if any errors occur, and also output an appropriate error message.

[15 marks]

(c) Write a C function called `writeCounts`, which:

- Imports:
  - A filename as a `char` pointer — the output file.
  - A pointer to an array of the same type returned by the `readProfiles` function from part (b).
  - The length of the array.
- Cycles through all the profiles to determine:
  - The name of the person who follows the **highest** number of people (or *a* person who follows the *equal*-highest number), along with the number followed.
  - The name of the person who follows the **lowest** number of people, along with the number followed.
- Opens the specified text file for writing.
- Writes the names and "following" counts to the file on two lines, with the highest first and the lowest second. Each line should begin with a name, followed by "=" and then the count, which should have a field width of 5. For example

```
Les=    3
Joe=    1
```

[10 marks]

(d) Write a `main` function in C which:

- Reads two filenames from the user — the input and output files.
- Uses the function `readProfiles` from part (b) to read the input file.
- Uses the function `writeCounts` from part (c) to write to the output file. [5 marks]

(e) Write the appropriate function prototype declarations (as they would appear in a header file) for the functions from parts (b) and (c). [2 marks]

—— End of Examination Paper ——