Question 1 (11 marks)

For each of the following type declarations:

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

(a) `enum Cosmic { STAR = 6, PLANET = 3, MOON, ASTEROID };` [2 marks]

(b) `typedef int const * const Quasar;` [2 marks]

(c) [3 marks]
```
typedef union {
        struct {
            int mercury;
            int venus;
        } earth;
        char* mars;
    } Jupiter;
```

(d) [4 marks]
```
struct Ceres {
        struct Ceres * pluto;
        struct Ceres ** makemake;
        void (* const eris)(char dysnomia);
    };
```

Question 2 (9 marks)

Write a short (but complete) C program to print a random number of random numbers. Your program should accept two command-line parameters. The first is the upper bound for each random number. The second is the upper bound for the number of numbers to print.

For instance, your program may be called as follows:

```
./randomlist 6 4
```

Given this command-line, your program should print *up to* 4 numbers, each of which lies in the range 0–6.

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.

```
float* a = (float*)malloc(3 * sizeof(float));
float* b = (float*)malloc(3 * sizeof(float));
float** c = (float**)malloc(2 * sizeof(float*));
float** d = (float**)malloc(2 * sizeof(float*));
int i;

for(i = 0; i < 3; i++) {
    *(a + i) = i;
    b[i] = 3 - i;
}

*(c + 0) = &b[(int)a[2]];
*(c + 1) = a + (int)b[2];
d[0] = c[1] + 1;
d[1] = b;

d[1][1] += c[1][1];
(*c)[0] += (*d)[0];
```

Based on this:

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

(b) Show the final contents of all arrays containing non-pointer values. [4 marks]

Question 4 (20 marks)

The following function (on the next page) implements a simple music player. The `play()` function receives a playlist from the calling function, and plays each track on the playlist sequentially. If any of the tracks cannot be played, it will print an error message and skip to the next track.

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

Question 4 continues on the next page

```
1  void play(PlayList* playlist) {
2      char* filename;
3
4      int eop = endOfPlayList(playList);
5      while(!eop)
6      {
7          filename = getNextSong(playList);
8          if(validFile(filename))
9          {
10             printf("Now playing %s\n", filename);
11             playTrack(filename);
12         }
13         else
14         {
15             printf("File cannot be read.\n");
16         }
17         eop = endOfPlayList(playList);
18     }
19 }
20
```

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 program works fine, except that any track names in upper case produce the error message "File cannot be read", even if they are valid tracks. [5 marks]

(b) Nothing happens at all — no messages and no audio output. [5 marks]

(c) The program reports that it is playing each track, but does not actually do so. [5 marks]

(d) The program works fine on any playlist containing only valid tracks. However, any invalid track will result in a segfault after the "Now playing…" message is output. [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 set of real numbers, each representing the exchange rate between the Australian Dollar and:

- the US Dollar,
- the Euro, and
- the Yen.

(ii) A collection of economic data for a given year, consisting of:

- the year in question,
- a pointer to the data type from part (a) containing the (average) exchange rates,
- the economic growth rate (a real number, possibly negative),
- the increase in unemployment compared to last year (a real number, possibly negative), and
- an array containing the increase in unemployment for each month of the year.

[8 marks]

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

- The first line of the file gives the total number of years of economic data (a single integer).
- Each subsequent line contains information on one year, separated by spaces, as follows:
  - the year,
  - three real numbers indicating the currency exchange rates (in the order given above),
  - the economic growth rate for the year, and
  - the rise in unemployment for the year.

For example (and these are *made-up* numbers):

```
4
2008 0.90 0.55 82.1 0.1 1.2
2009 0.95 0.60 95.3 1.5 -0.1
2010 1.01 0.65 101.2 2.3 -0.3
2011 1.04 0.67 99.3 2.2 0.2
```

The monthly rise in unemployment is estimated by dividing the yearly value by 12. For instance, in 2008, the yearly rise in unemployment is 1.2%, so we estimate the monthly rise to be 0.1%. This will be the same for every month of the year.

Question 5 part b continues on the next page

Your `readStats` 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 datatypes from part (a).
- Read the data from the file into the array, filling in the monthly unemployment figures 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 `writeResults`, which:

- Imports:
  - A filename as a `char` pointer — the output file.
  - A pointer to an array of the same type returned by the `readStats` function from part (b).
  - The length of that array.
- Cycles through all profiles to determine:
  - the highest economic growth on record, and the year it occurred, and
  - the lowest rise in unemployment on record, and the year it occurred.
- Opens the specified text file for writing.
- Writes the results to the file. The highest economic growth should appear on the first line, and the lowest rise in unemployment on the second. Both numbers should have two decimal places, and a field width of five. The years should appear in parentheses after each value. For example:

```
 2.30 (2010)
-0.30 (2010)
```

[10 marks]

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

- Reads two filenames from the user via standard input — the input and output files.
- Uses the function `readStats` from part (b) to read the input file.
- Uses the function `writeResults` 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 Supplementary Examination Paper ——