UCP120_20132

Curtin University

Department of Computing
End of Semester 2 Central Examinations - November 2013

Attendance Mode:	Internal
Centre(s):	Bentley Campus
Unit(s):	**10163 - Unix and C Programming 120**
Duration:	**2 Hours** *Prior to commencement of the examination there will be a 10-minute reading period. During this period notes may be written **in margins or reverse of the examination paper**. Commencement of the examination will be indicated by the supervisor.*
Total Marks:	100
Calculator:	No, not allowed

Supplied by the University:

1 x 16 page answer book

Supplied by the Student:

None

THIS IS A CLOSED BOOK EXAMINATION

Other Information:

The exam consists of FIVE (5) questions.
ANSWER ALL QUESTIONS.
The marks allocated for each question are shown beside the question.

Examination paper is to be released to student

Question 1 (11 marks)

Each of the following descriptions represents a datatype. For each one, using C code:

(i) Declare the datatype.
(ii) Declare one variable having that datatype (not a pointer to it).
(iii) Initialise the variable. (The actual values are your choice. The simplest possible initialisation code will suffice, but you must *completely* initialise the variable.)

Choose any valid names, where names have not already been given.

(a) An enum called `Breakfast` with the possible values `eggs`, `bacon` and `beans`, having corresponding integer values 0, 5 and 10. **[2 marks]**

(b) A struct containing (1) a 2×2 array of real numbers, and (2) a pointer to a function that takes no parameters and returns an integer. **[3 marks]**

(c) A linked list node struct, where the linked list stores pointers to the struct from part (b). **[3 marks]**

(d) A union that can store either (1) an integer, (2) a pointer to a constant real number, or (3) a struct containing both an integer and a pointer to a constant real number. **[3 marks]**

Question 2 (9 marks)

Write a C function called `sillyRandom`, taking two integer pointer parameters and returning nothing.

The function should:

- Set the first parameter to the sum of both original values.
- Set the second parameter to a random number between the previous sum (as calculated on the previous call to `sillyRandom`) and this sum, inclusive. On the first execution, consider the "previous sum" to be zero.

Note: the current sum may be smaller, larger or equal to the previous sum.

Question 3 appears on the next page

Question 3 (20 marks)

Consider the following code:

```
double **rimmer[2];
double **kryten[2];
double *cat[3];
double lister[] = {4.0, 7.0, 11.0};

rimmer[0] = (double**)malloc(2 * sizeof(double*));
rimmer[1] = (double**)malloc(2 * sizeof(double*));
*kryten = *(rimmer + 1);
kryten[1] = cat;

**rimmer = lister;
(*rimmer)[1] = (*rimmer)[0];
*(rimmer[1]) = lister + 1;
rimmer[1][1] = (double*)malloc(3 * sizeof(double));
kryten[1][0] = &rimmer[1][1][0];
kryten[1][1] = rimmer[1][1] + (int)(***rimmer * 0.5);

rimmer[1][1][0] = kryten[0][0][1];
kryten[0][1][1] = rimmer[0][0][1];
```

Based on this:

(a) Draw a diagram showing all the pointer relationships created. **[15 marks]**

(b) Show the contents of all non-pointer values at the end. Indicate any that are left uninitialised. **[5 marks]**

Question 4 (20 marks)

The following function (on the next page) controls the floodgates for a dam (i.e. doors that are opened to release water from the lake behind the dam).

If the water level in the lake is too high, or if the level will soon become too high based on forecast rainfall, the dam's floodgates should be opened to lower the level.

The function imports the number of floodgates, and a pointer to a `Dam` struct that stores information on the dam. It calls other functions to help decide how many floodgates need to be opened.

You believe the `operateFloodgates` function itself is working. However, you suspect that the functions it calls may have defects.

Question 4 continues on the next page

```c
void operateFloodgates(int nGates, Dam *theDam)
{
    int *forecast;
    int day, nDays, gate, nOpenGatesNeeded;
    double waterLevel = damLevel(theDam);

    forecast = getRainForecast(&nDays);
    for(day = 0; day < nDays; day++)
    {
        waterLevel += calcWaterIntake(theDam, forecast[day]);
    }
    free(forecast);
    printf("Forecast water level = %lf\n", waterLevel);

    nOpenGatesNeeded = calcOpenGatesNeeded(theDam, waterLevel);
    if(nOpenGatesNeeded > nGates)
    {
        nOpenGatesNeeded = nGates;
        printf("Catastrophe imminent. Enact evacuation procedures.\n");
    }

    for(gate = 0; gate < nOpenGatesNeeded; gate++)
    {
        openGate(theDam, gate);
    }
}
```

For each situation below:

(i) Give **two plausible hypotheses** for what might be wrong (in the code *not* shown).
(ii) How do the hypotheses fit the observations?
(iii) What debugging steps (e.g. breakpoints, monitoring of particular variables) will help narrow down the problem?
(iv) How will you know which hypothesis is correct (or more likely to be correct)?

State any relevant (and realistic) assumptions you make about the code not shown.

(a) The function outputs the correct forecast water level, but never opens any gates, even when the level requires it. **[5 marks]**

(b) The forecast water level output by the function is always exactly the current level, regardless of any rain forecast. **[5 marks]**

(c) The function segfaults without printing anything. **[5 marks]**

(d) If the forecast is for no rain, the function works correctly. If rain is forecast, the function segfaults after reporting the correct forecast water level. **[5 marks]**

Question 5 appears on the next page

Question 5 (40 marks)

For this question, refer to the following standard C function prototypes:

```
FILE *fopen(const char *path, const char *mode);
int fclose(FILE *fp);
int fscanf(FILE *stream, const char *format, ...);
char *fgets(char *s, int size, FILE *stream);
int feof(FILE *stream);
int strcmp(const char *s1, const char *s2);
char *strcpy(char *dest, const char *src);
```

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

(i) A record of land use over an area, including (1) the number of people living there, and (2) the type of land use: urban, agricultural or wilderness, as a single character. **[2 marks]**

(ii) A 2D rectangle, consisting of an integer width and length. **[2 marks]**

(iii) A collection of land-use records. This contains:

- A description, up to 127 characters.
- A date – a sequence of 8 digits in string form.
- A 2D grid of the datatype described in part (i). The grid can have any width and length.
- The width and length in metres of the grid cells (i.e. each plot of land represented by the datatype in part (i)). All grid cells have the same width and length.

Use the datatypes from part (i) and (ii) where applicable.

[4 marks]

Question 5 continues on the next page

(b) Write a C function called `loadLand` to read a collection of land-use data from a text file.

The first four lines of the file contain the following:

Line 1:	The description of the data.
Line 2:	The date of the data.
Line 3:	The width and length of each grid cell, in metres, separated by "`x`" (e.g. "`150x200`").
Line 4:	The width and length of the whole grid, in the same format. We'll call these values W and L. ($W \times L$ gives the total number of grid cells.)

The subsequent lines contain the land use information itself. There are L lines, each containing W land use records, separated by a space.

Each record consists of an integer – the number of people – followed immediately by a "`U`" (for urban), "`A`" (agricultural) or "`W`" (wilderness). For example, "`500U`" indicates 500 people in an urban environment.

Here's an example file:

```
A small town
20130510
200x200
5x4
0W 3A 8A 4A 0W
0W 4A 300U 20A 1A
1W 30A 500U 100U 10A
0W 2W 150U 50U 15A
```

Your function should:

- Take in a filename parameter.
- Read the file, according to the above specifications.
- Dynamically allocate the structures you designed in part (a).
- Store the file data in these structures.
- Return a pointer to the main structure from part (a)(iii).

If the file *cannot* be opened, your function must return `NULL`. An error message is not required. If the file *can* be opened, you may assume that it definitely conforms to the specification.

[17 marks]

Question 5 continues on the next page

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

- Imports:
  - A pointer to the main struct from part (a)(iii) (the same type *exported* by the `loadLand` function).
  - A pointer to a real number, called `popDensity`.
  - A pointer to a character, called `urban`.
- Returns nothing.
- Calculates the average population density – the total number of people divided by the total land area (across all grid cells). This should be exported via the `popDensity` parameter. (Note: unit conversion is not required.)
- Determines the number of urban grid cells, to be exported via the `urban` parameter.

[10 marks]

(d) Write a `main` function that accepts (and requires) one filename as a command-line parameter.

Your `main` should:

- Use the function `loadLand` from part (b) to read the input file.
- Use the function `analyseLand` from part (c) to calculate population density and urban land usage.
- Output the results, or (if required) any error messages.
- Perform all necessary cleaning up.

[5 marks]

— End of Examination Paper —