UCP 120 - 20142.

Curtin University

Department of Computing
End of Semester 2 Central Examinations - November 2014

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

IMPORTANT INFORMATION

- The possession or use of:

Mobile phones or any other device capable of communicating information, are prohibited during examinations.

Electronic Organisers/PDAs (with the exception of calculators) or other similar devices capable of storing text or restricted information are prohibited during examinations.

Any breach of examination regulations will be considered cheating and appropriate action will be taken in accordance with University policy.

Other Information:

This exam contains FIVE (5) questions. Answer all questions in the answer book provided.

The marks allocated for each question are shown beside the question.

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) A struct containing (1) string of indefinite length, and (2) a pointer to a function taking a string and returning a string. **[3 marks]**

(b) A union that can store either (1) an array of 4 void pointers, or (2) an enum with possible values of "plant", "animal" or "fungus", having corresponding integer values 3, 2 and 1. **[3 marks]**

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

(d) A pointer to a function, taking in a variable-length array of the struct from part (a) and returning nothing. **[2 marks]**

Question 2 (9 marks)

Write a C function called `sumRandom` that randomly selects elements from an array and adds them up. Each random selection is performed independently, meaning that some array elements may (by chance) be chosen more than once.

The function should take four parameters: an array of real numbers and its length, the number of elements (n) to add, and a pointer to a real number so as to export the sum. The function should return nothing.

If $n < 0$, the function should select $|n|$ array elements and export the inverse of the sum ($-$sum). If $n = 0$, the function should export 0.0 as the sum.

Question 3 appears on the next page

Question 3 (20 marks)

Consider the following code:

```
int i;
char **x, **y, **p, **q, **r;
char s[] = "Hello";
char t[] = "World";

x = (char**)malloc(5 * sizeof(char*));
y = (char**)malloc(5 * sizeof(char*));
for(i = 0; i < 5; i++)
{
    *(x + i) = s + i + 1;
    *(y + 4 - i) = t + (i / 2);
}

p = x;
q = &y[2];
**p = q[0][1];
r = p;
p = q + 1;
q = r + 1;
**p = q[0][1];

free(x);
free(y);
printf("%s %s\n", s, t);
```

Based on this:

(a) Draw a diagram showing all the pointer relationships created. If any pointers change, *neatly* and *clearly* cross-out the old arrows. **[16 marks]**

(b) Show the output. **[4 marks]**

Question 4 appears on the next page

Question 4 (20 marks)

The following function is a (very simplistic) aircraft autopilot system. It takes a destination, and uses various functions to figure out how to get there, and to keep the pilots informed. The autopilot is engaged *after* take-off, and disengages automatically *before* landing.

You believe the `autopilot` function itself is working, but you suspect that the functions it relies on may have defects.

```
1  void autopilot(void *destination)
2  {
3      int destReached;
4      double lat, long, altitude, targetAlt, direction, targetDir;
5      do
6      {
7          gps(&lat, &long, &altitude);
8          getHeading(&direction);
9          navReadout(altitude, direction);
10         navigate(destination, lat, long,
11                  &targetAlt, &targetDir, &destReached);
12
13         if(!destReached)
14         {
15             if(altitude < (targetAlt - ERROR_MARGIN))
16             {
17                 climb();
18                 indicatorLight(CLIMBING);
19             }
20             else if(altitude > (targetAlt + ERROR_MARGIN))
21             {
22                 descend();
23                 indicatorLight(DESCENDING);
24             }
25             else
26                 indicatorLight(OFF);
27
28             adjustHeading(targetDir - direction);
29         }
30
31         wait(1); /* Do nothing for 1 second */
32     }
33     while(!destReached);
34 }
```

Question 4 continues on the next page

Fortunately, you can use a simulator to test the code on the previous page without endangering actual lives or aircraft.

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 (simulated) aircraft simply flies around in circles. **[5 marks]**

(b) The aircraft inevitably dives and crashes into the ground as soon as the autopilot comes on. **[5 marks]**

(c) The autopilot segfaults as soon as the aircraft's altitude is too low. **[5 marks]**

(d) The autopilot segfaults after displaying the current altitude and direction. **[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, ...);
int fgetc(FILE *stream);
char *fgets(char *s, int size, FILE *stream);

int sscanf(const char *str, const char *format, ...);
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 TV show, including the time it begins, the time it ends, the weekday on which it is shown (Monday–Sunday), and the name, having up to 100 characters. The start time and end time each have separate hour and minute components. **[3 marks]**

(ii) The weekly schedule for a TV channel, consisting of the name of the channel (one word, up to 10 characters long), and a collection of TV shows. **[3 marks]**

(iii) A TV guide, consisting of collection of channels and their scheduled shows. **[2 marks]**

Question 5 continues on the next page

(b) Write a C function called `readTV` to read TV show schedules from a text file.

The first line of the file contains the following information in order:

- The number of TV channels.
- The name of the 1st channel (one word).
- The number of shows scheduled on the 1st channel.
- The name of the 2nd channel (one word).
- The number of shows scheduled on the 2nd channel.
- etc.

The number of subsequent lines can be determined from this information. Each subsequent line contains details for a single TV show; specifically:

- The start time in hh:mm format (e.g. "18:30").
- The end time in hh:mm format.
- A three-letter lowercase abbreviation indicating the day on which the show starts ("mon", "tue", "wed", etc.)
- The name of the show.

All information is separated by whitespace, and the name of the show may also contain whitespace.

Shows are listed in order of channel. Thus, if the 1st channel has 5 shows and the 2nd has 7, then lines 2–6 will represent shows on the first channel, and lines 6–12 represent shows on the 2nd channel.

Here is an example file:

```
3 ABC 2 Seven 1 BBC 2
20:30 20:55 mon The Business
17:35 17:50 fri The High Fructose Adventures Of Annoying Orange
9:00 11:30 thu The Morning Show
9:05 10:05 tue Natural World
19:00 19:30 wed World News Today
```

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 `whatsOn`, which:

- Imports:
  - A pointer to the main struct from part (a)(iii) (the same type *exported* by the `readTV` function).
  - A string containing a 3-letter lowercase abbreviation of a weekday.
  - A string containing a time in hh:mm format.
- Returns nothing.
- Determines whether any TV shows are on at the time given; i.e. they are scheduled on the given weekday, start at or before the time given, and finish at or after the time given. (You may assume all shows start and end on the same day.)
- Outputs the channel name and show name for each matching show.

[10 marks]

(d) Write a `main` function that accepts (and requires) three command-line parameters: a filename, a weekday abbreviation, and a time in hh:mm format.

Your `main` should:

- Use the function `readTV` from part (b) to read the input file.
- Use the function `whatsOn` from part (c) to find and display TV shows on at the given time.
- Output any error messages (if required).
- Perform all necessary cleaning up.

[5 marks]

End of Examination Paper