Curtin University

Department of Computing
End of Semester 2 Supplementary/Deferred Examination - February 2013

Attendance Mode:	Internal	
Centre(s):	Bentley Campus	
Unit(s):	**10163 - Unix and C Programming 120** **10225 - Unix and C Programming 500**	
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:

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

Question 1 (11 marks)

For each of the following type declarations:

- Give an example of **declaring and initialising a single variable** of that type.
- State what types of values that variable can hold.
- If applicable, state how the various components of the variable are arranged in memory.

(a) **[2 marks]**

```
union Alpha {
    int beta;
    int *gamma;
};
```

(b) **[3 marks]**

```
typedef struct Delta {
    const struct Delta *epsilon;
    int (*zeta)();
} Delta;
```

(c) **[2 marks]**

```
typedef enum {THETA, IOTA, KAPPA, LAMBDA} Eta;
```

(d) **[2 marks]**

```
struct Mu {
    int xi[RHO][SIGMA];
    struct {
        double *pi;
    } omicron;
};
```

(e) **[2 marks]**

```
typedef union {
    struct {
        int tau;
        void *upsilon;
    } phi;
    char *chi;
} Nu;
```

Question 2 (9 marks)

Write a C function called `myRand`, which takes two parameters: an integer and a pointer to an integer. Your function should generate a random number, exporting it via the pointer parameter.

Generated random numbers must fall within a particular range. The first parameter (passed by value) represents the maximum value. The minimum value is the *previous* maximum value; that is, the maximum value used on the previous call to `myRand`. On the first call, the minimum value is zero.

For example, consider the following calls:

```
myRand(10, &a);
myRand(20, &b);
```

The first call above should store a random number between 0 and 10 in `a`. The second call should store a random number between 10 and 20 in `b`.

Question 3 appears on the next page

Question 3 (20 marks)

Consider the following code:

```
int **a[4], i;

a[0] = (int**)malloc(2 * sizeof(int));
a[1] = (int**)malloc(3 * sizeof(int));
a[2] = (int**)malloc(sizeof(int));
a[3] = a[1] + 1;
a[3][1] = (int*)malloc(6 * sizeof(int));

*(a[1] + 1) = *(a[1] + 2) + 1;
*(a[1]) = *(a[3]) + 1;
a[2][0] = *(a[1]) + 1;
a[0][1] = *(a[2]) + 1;
**a = (*a)[1] + 1;

for(i = 0; i < 6; i++)
{
    a[3][1][i] = i;
}

for(i = 0; i < 4; i++)
{
    **a[i] *= i;
}
```

Based on this:

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

(b) Show all non-pointer values (except for `i`) at the end. **[5 marks]**

Question 4 (20 marks)

The following function (on the next page) implements an algorithm to control an ice-cream-testing robot. The robot is able to pick up an open ice cream container, take samples and analyse them to determine whether the product is up-to-standard. Several samples must be taken and analysed. If any of the samples are below standard, the test fails. The function's return value indicates whether the test passed or failed.

The `iceCreamQA` function itself is defect-free, but there are defects in the other functions it calls (not shown here).

Question 4 continues on the next page

```
1  int iceCreamQA(int numSamples)
2  {
3      IceCreamRobot *robot = connectToRobot();
4      int i = 0, success = TRUE;
5
6      if(acquireIceCream(robot))
7      {
8          while(success && i < numSamples)
9          {
10             printf("Analysing sample...\n");
11             if(getSample(robot))
12             {
13                 analyse(robot, &success);
14                 i++;
15             }
16             else {
17                 success = FALSE;
18             }
19         }
20     }
21     else
22     {
23         printf("Unable to test ice cream.\n");
24         success = FALSE
25     }
26
27     return success;
28 }
```

For each situation below:

- Give **two plausible hypotheses** for what might be wrong (in the code *not* shown). Briefly explain why both hypotheses fit the observations.
- Explain how you would use debugger features to rule one of them out. In particular, where would you set **breakpoints**, and which **variables** would you monitor?

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

(a) The function outputs "Unable to test ice cream" when everything should be working. **[5 marks]**

(b) The function outputs "Analysing sample…" several times but always returns FALSE. **[5 marks]**

(c) The function outputs nothing and segfaults. **[5 marks]**

(d) The function outputs "Analysing sample…" once only, then hangs — continuing to run but without doing anything. **[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.

You may 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);
int atoi(const char *nptr);
```

(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 employee payment details, including: an hourly pay rate, and the number of hours worked in a week (both real numbers). **[2 marks]**

(ii) An employee record, where each employee has a name (up to 100 characters), an employee number (an integer), their manager's employee number, and a set of their payment details. **[3 marks]**

(iii) A set of employee records, able to hold the details of any number of employees. **[3 marks]**

Question 5 continues on the next page

(b) Write a C function called `loadPeople` to read a list of personnel records from a text file.

The first line of the file indicates how many records follow. Each subsequent line contains one record, consisting of a name, employee number, manager employee number, pay rate and hours per week, in that order, separated by spaces.

Spaces in an employee's name are indicated by underscore ("_") characters rather than actual spaces.

The following is an example file:

```
3
Alex_Smith 445 1 35.67 37.5
Big_Bird 1 0 100 16.75
Donald_J_Brown 789 445 4500 0.5
```

Your function should:

- Import a filename.
- Read the file, according to the above specifications.
- Dynamically allocate the structures you designed in part (a).
- Store the command sequence in these structures.
- Return a pointer to a `Drawing`, or `NULL` if an error occurs.

If the file can be opened, you may assume that it definitely conforms to the specification.

[17 marks]

(c) Write a C function called `calcPay`, which determines the average weekly income of all employees having a particular manager.

Specifically, the function must:

- Import a pointer to the information returned by the `loadPeople` function.
- Import an employee number for the manager in question.
- Return nothing.
- Cycle through the employees, finding those whose manager is the manager in question. (Consider only those *directly* working for the manager in question.)
- Output the name and weekly income for each of those particular employees.
- Output the overall average weekly income of those employees.

All output numeric values should have two decimal places and a field width of ten characters.

[10 marks]

Question 5 continues on the next page

(d) Write a `main` function that accepts two command-line parameters: an input filename and an employee number.

Given these, `main` should:

- Conduct all appropriate error checking and handling.
- Use the function `loadPeople` from part (b) to read the input file.
- Use the function `calcPay` from part (c) to calculate and display the average weekly pay.
- Perform all necessary cleaning up. **[5 marks]**

— End of Supplementary/Deferred Examination Paper —