End of Semester 1, 2019
Exam
COMP1000 Unix & C Programming

Venue	____________________
Student Number	|__|__|__|__|__|__|__|__|
Family Name	____________________
First Name	____________________

Curtin University

School of Electrical Engineering, Computing and Mathematical Sciences

EXAMINATION

End of Semester 1, 2019
Exam

COMP1000 Unix & C Programming

This paper is for Bentley Campus and Miri Sarawak Campus students

This is a CLOSED BOOK examination

Examination paper IS NOT to be released to student

Examination Duration 2 hours

Reading Time 10 minutes

Total Marks 100

Supplied by the University

None

Supplied by the Student

Materials

None

Calculator

No calculators are permitted in this exam

Instructions to Students

This exam contained FIVE (5) questions. Answer all questions in the space provided.

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

For Examiner Use Only

Q	Mark
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	

Total __________

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 Snake with the possible values Hognose, Bull, Carpet, and Garter, having corresponding integer values 4, 1, 2 and 5. **[2 marks]**

(b) A struct containing (1) a 2 by 3 array of real numbers, and (2) a pointer to a function that takes an integer number and returns a string. **[3 marks]**

(c) A self-referential struct, containing three pointers (called left, right, and parent) to other instances of the same struct, as well as a pointer to a 1d array, the type of which can change. **[3 marks]**

(d) A union that can store either (1) the enum in part (a), (2) the struct in part (b), or (3) a single character. **[3 marks]**

Question 2 appears on the next page

Question 2 (14 marks)

Write a C function called evenOdd, which takes a pointer to an integer number **a** and actual integers **b** and **c**. The function should return an actual integer.

The function should generate a random number between b and c, and then export (via a) the closest even number (either the number itself or the previous number), while returning the closest odd number (either the number itself or the next number).

For completeness, you must seed the random number generator in your function.

Your algorithm MUST NOT use modulo, if statements, case statements, ternary operators, or any other form of selection control.

Hint: Use bitwise operators.

Question 3 appears on the next page

Question 3 (12 marks)

(a) How would you display all of the files in the "UCP" directory and sub-directories that start with 2 underscores, followed by any number of any characters, ending in a number, with the ".c" extension.

Hint: use the find command. **[6 marks]**

(b) For the following UNIX shell script, use a meaningful example input to explain the purpose and any possible output.

```
str=""
for word in $*; do
    if [ $word == "Snake" ]; then
        str="Danger Noodle $str"
    else
        str="$word $str"
    fi
done
echo $str
```

[6 marks]

Question 4 appears on the next page

Question 4 (15 marks)

The following function implements a character builder for a game/story. It adds a new character to the existing character list and requests all the character details(name, age, biography, skills) from the user. It also links the new character to any characters it is associated with. This is done by a name search on the existing character list.

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

```
int createCharacter()
{
    Character* npc;
    char* friend;
    int n,ii;

    /*add a new character to game*/
    npc = addCharacter();

    /*populate all the details of the npc,
    including their name and abilities*/
    fillDetails(npc);

    printf(”How many friends does this character have?\n”);
    scanf(”%d”, &n);

    for(ii = 0; ii < n; ii++)
    {
        /*reads the name of the friend from the user,
        store it in a dynamically allocated string*/
        friend = readName();

        /*finds the character struct with the same name as
        the entered friend and links the two structs together
        in their friend lists.*/
        linkFriend(friend, npc);
        free(friend)
    }

    /*prints all the details of the character to the user*/
    printCharacterDetails(npc);
}
```

Question 4 continues on the next page

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)?

(a) A segmentation fault occurs before the user is prompted to enter any data. **[5 marks]**

(b) No friends can ever be matched/linked, even when it is know that the character exists in the list. **[5 marks]**

(c) A segmentation fault occurs after the user enters a friends' name but before they are prompted for the next name. **[5 marks]**

Question 5 appears on the next page

Question 5 (48 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 *strncpy(char *dest, const char *src, size_t n);
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 food item, including the name (a string of up to 100 characters), the quantity (a real number), and the unit type (a 6 letter character code). **[2 marks]**

(ii) A list of food items of unknown size.
Hint: Marks have been allocated for using a linked list. **[6 marks]**

(iii) A cooking plan structure consisting of a list of all the food items in the pantry, and a list of all the food items in a recipe. **[5 marks]**

Question 5 continues on the next page

(b) Write a C function called `loadFood` to read a list of recipe ingredients and pantry items from a text file.

The first part of the file contains an unknown number of lines. Each line describes the next ingredient, consisting of a quantity, then a space, then the unit of measurement, then a space, and then the name of the item, which takes up the rest of the lines and may include additional spaces.

After the recipe ingredients the next line of the file is `---`. The lines after these three hyphens contain the food items currently in the pantry. They follow the same format as the recipe items.

The following is an example file:

```
200 grams butter
0.75 cup brown sugar
1 cup plain flour
0.25 cup cocoa
2 cup cornflakes
2 single weetbix
0.25 cup hazelnuts
---
0.5 box cornflakes
0.5 box cocoa
1 kg s.r. slour
1 litre milk
0.75 pack almonds
1 kg brown sugar
```

In this example, there are seven recipe ingredients and 6 pantry items.

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. **[16 marks]**

Continue answer for Question 5 on the next page

Continue answer for Question 5 on the next page

(c) Write a C function called `generateShopping` that generates a shopping list based on what you need and what you currently have. To complete this function you will need to make use of the existing helper function `checkAmount`, that takes two pointers to the structure defined in part a(i), the first being the recipe item and the second being the pantry item. It will then return true (1) if there is enough of the item in the pantry for the recipe, and false (0) if more is required.

You **DO NOT** need to write the function `checkAmount`!

The function `generateShopping` must must:

- Import a pointer to the information returned by the `loadFood` function.
- For each recipe item, check whether a similar item is in the pantry:
  - If no match is found then add that item to the shopping list.
  - If a matching item is found then use the `checkAmount` function to determine if more is required and if so add it to the shopping list. If there is enough of that item in the pantry for the recipe then it does not need to be added to the list.
- Once you have assembled the shopping list, output the name of each item to the user. The list should appear with one item per line. The quantity is not required.

Hint: use the structure from part a(ii) to store the shopping list.

Given the example file shown in part (b), your function should output this:

```
Please add these items to your shopping list:
    butter
    plain flour
    weetbix
    hazelnuts
```

[14 marks]

Continue answer for Question 5 on the next page

Continue answer for Question 5 on the next page

(d) Write a `main` function that accepts (and requires) *one or more* filenames as command-line parameters.

Your `main` should:

- Conduct all appropriate error checking and handling.
- Perform all necessary cleaning up.
- For each filename:
  - Use the function `loadFood` from part (b) to read the file.
  - Use the function `generateShopping` from part (c) to generate the individual shopping list for each recipe.

[5 marks]

End of Examination