Curtin University — Department of Computing

UNIX and C Programming (COMP1000)

Mock Examination

Practice for Final Exam

Real exam weight: 50% of the unit mark.
To pass the unit, your unit mark must be ≥50%, AND your exam mark must be ≥45%.

Attempt this mock exam in preparation for the final exam. Answer all questions by yourself. In the real exam, you will have 120 minutes, plus 10 minutes reading time.

Will the real exam be like this?

The real exam will follow approximately the same format and will cover the same material. However, the questions will be different (so trying to memorise answers will get you nowhere!) The real exam will also be closed book – no books, notes, electronic devices, etc.

How can I get help/feedback?

First, make your best attempt. Then, to obtain feedback, see your tutor, or the senior tutor, or the lecturer.

Will you upload the answers?

No. Sample answers to this mock exam *will not* be provided – no exceptions.

Why?

Sample answers *discourage* people from putting in real effort to learn the concepts and skills. They encourage rote (fake) learning, where you try to memorise an answer without understanding how to obtain it or even why it's correct.

Basically, if you're given the answers, it's too easy to convince yourself that you don't need to work them out.

Updated: 4th November, 2015

Question 1 (10 marks)

For each of the following type declarations:

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

(a)
```c
typedef enum {ALPHA, BETA = 3, GAMMA, DELTA = 6} Greek;
```

(b)
```c
typedef struct Other {
    double x;
    int* y[2];
    struct Other* z;
} Other;
```

(c)
```c
typedef union {
    double * const * ptr;
    int val;
    Other o;
} Misc;
```

(d)
```c
struct Multi {
    int which;
    union {
        float f;
        double d;
        int i;
    } data;
};
```

(e)
```c
typedef int const * (* const CallMe)(const int *);
```

Question 2 appears on the next page

Question 2 (10 marks)

For each of the following UNIX shell scripts, explain what is output:

(a)
```
for file in $*; do
    for word in $(cat $file); do
        if [ $word == Fungible ]; then
            echo 1
        fi
    done
done
```

(b)
```
echo Enter name:
read name
if ! [ -x $name ]; then
    chmod 755 $name
fi
./$name
```

(c)
```
while ./myprogram; do
    ./myotherprogram
done
```

(d)
```
grep -E 'X[A-Z]*X' $1
```

(e)
```
grep -E '[0-4]+5?(6|7)' $1
```

(f)
```
grep -E '^The.*end\$' $1
```

Question 3 appears on the next page

Question 3 (10 marks)

(a) Write a short C program to output its command-line parameters in reverse order. That is, when called like this:

```
./program one two three four
```

your program should output:

```
four
three
two
one
```

(b) Write a C function to return a random power of 2, between 2^0 and 2^{30}, inclusive.

Your function should take no parameters and return an `int`. The return value should be one of 2^0, 2^1, 2^2, etc. 2^{30}, chosen with equal probability.

(You do not need to seed the random number generator.)

Question 4 appears on the next page

Question 4 (20 marks)

Consider the following code.

```
float a[] = {0.1, 0.2, 0.3};
float b[] = {1.0, 2.0, 3.0};
float c[] = {10.0, 20.0, 30.0};

float** x = (float**)malloc(3 * sizeof(float*));
float** y = (float**)malloc(3 * sizeof(float*));
float** p = x + 1;
float** q = &y[2];

*x = a;
*y = b;
*p = c;
*q = p[0] + 2;
x[1] = &x[0][1];
*(y + 1) = x[1] + 1;
p[1] = &y[0][(int)(*q)[0] % 7];

**x = **p + **y;
y[1][0] = p[1][0] + x[1][0];
*q[0] = (*y)[1] + *(*p + 1);
```

Based on this:

(a) Draw a diagram showing all the pointer relationships created.

(b) Show the contents of **a**, **b** and **c** at the end.

Question 5 appears on the next page

Question 5 (20 marks)

The following code implements a search-and-replace feature in a text editor. The user enters two strings. The function finds each occurrence of the first ("search") string and, if the user agrees, replaces it with the second ("replacement") string.

For each occurrence, the user is shown a small snippet of surrounding text, with the matching text highlighted. The user is asked whether to replace each occurrence or leave it unchanged. The program then finds the next occurrence, if there is one.

However, there are defects — not in the code shown below but in the functions it calls.

```
1  void searchAndReplace(char* text)
2  {
3      char search[MAX_LENGTH + 1];
4      char replacement[MAX_LENGTH + 1];
5      int found, location = 0, len;
6      int totalLength = strlen(text);
7
8      /* Ask user for the term to search for and its replacement.
9         (The user can also cancel the operation here.) */
10     if(readSRInput(search, replacement, MAX_LENGTH))
11     {
12         len = strlen(search);
13
14         do /* Loop until no more matches, or the end of text. */
15         {
16             /* Find the next match. */
17             found = findNext(text, search, &location);
18             if(found)
19             {
20                 printf("Found %s at %d\n", search, location);
21
22                 /* Ask the user whether to replace. */
23                 if(askToReplace(text, location, len))
24                 {
25                     /* Perform the replacement. */
26                     replace(text, location, len, replacement);
27                     location += strlen(replacement);
28                 }
29                 else
30                     location += len;
31             }
32         }
33         while(found && location < totalLength);
34     }
35 }
```

Question 5 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)?

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

(a) A segmentation fault occurs immediately after the user enters the search and replace strings, before anything else happens.

(b) No segmentation faults occur, but the program fails to find any text matching the search string, even when it definitely exists.

(c) The program appears to find text matching the replacement string, rather than the search string.

(d) All the occurrences of the search string are found. However, when the user is asked whether to replace each occurrence, the highlighting is one character off. That is, the text to be replaced is shifted one character to the right of what actually should be replaced.

(e) A segmentation fault occurs immediately after the user *confirms* that an occurrence of the search string should be replaced.

(f) The program replaces occurrences of the search string only when the user tells it not to.

Question 6 appears on the next page

Question 6 (30 marks)

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

```c
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 company, described by:

- A 3-letter code.
- The current share price (a positive real number).
- The total number of shares (a positive integer).
- Total asset value (a positive real number).
- Total debts (a positive real number).

(ii) A consortium of companies, including:

- A set of core member companies.
- A set of associate member companies.

Question 6 continues on the next page

(b) Write a C function called **`readConsortium`**, which:

- Imports a filename as a `char` pointer — the input file. This is a text file, structured as follows:
  - The first line contains two integers, separated by a space — the number of core members (c) and the number of associate members (a).
  - There are $c + a$ subsequent lines, each with the same format. The first c of these lines represent core companies and the last a lines represent associate companies.
  - Each line contains a 3-letter code, a share price (a real number), a number of shares (an integer), total asset value (a real number) and total debts (a real number), separated by spaces.

  For example:

```
3 2
AAB 0.54 1500000 5547569.40 99444.25
CGH 17.1 1000000 9837373.30 128585.05
XYZ 1.22 8000000 34484853.10 4004435.90
JMZ 80.8 7500000 48934734.40 8867434.65
IOP 0.25 15000000 8395754.75 10948.10
```

- Opens the file for reading.
- Dynamically allocates the appropriate memory for the required data structures from part (a).
- Reads the data from the file into the data structures.
- Returns a pointer to the data structure described in part (a) (ii).
- If an error occurs, return **`NULL`** instead; no error message is needed. You may assume that, if the file exists, it can definitely be read and will be in the correct format.

Question 6 continues on the next page

(c) Write a C function called **writeNetWorth**, which:

- Imports:
  - A filename as a `char` pointer — the output file.
  - A pointer of the same type returned by the **readConsortium** function from part (b).
- Opens the output file for writing.
- Cycles through all the companies in the consortium (both core and associate) and calculates their net worth, using the following formula:

$$\text{Net worth} = (\text{Share price} \times \text{Number of shares}) + \text{Asset value} - \text{Debts}$$

- Writes each value to the output file, with one line per company. Each output line should contain the 3-letter code, followed by a colon and then the company's net worth, with 2 decimal places and a field width of 12. For example:

```
AAB:  6258125.15
CGH: 26808788.25
XYZ: 40240417.20
JMZ:646067299.75
IOP: 12134806.65
```

(d) Write a **main** function in C, which:

- Reads two filenames from the user — the input and output files.
- Uses the function **readConsortium** from part (b) to read the input file.
- Uses the function **writeNetWorth** from part (c) to write the results to the output file.
- Outputs any necessary error messages.
- Performs any necessary cleaning up.

(e) Write a second **main** function in C, which:

- Accepts any number of command-line parameters, each representing an input file.
- Reads each input file using **readConsortium** from part (b).
- Writes output, using **writeNetWorth** from part (c), to a file whose name is "[input-file].out"; e.g. if the input file is called "data.txt", then the output file will be "data.txt.out".
- Outputs any necessary error messages.
- Performs any necessary cleaning up.

End of Mock Examination