Curtin University — Department of Computing

UNIX and C Programming (COMP1000)

Mock Test 2B

Practice for Test 2

Real test weight: 15% of the unit mark.

Attempt this mock test in preparation for Test 2. Answer all questions by yourself.

Will the real test be like this?

The real test 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 test 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 test *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: 6th October, 2015

Question 1

For each of the following descriptions:

(i) Write a suitable type declaration (or set of type declarations) for representing the data.

(ii) Show how to dynamically allocate the necessary memory, and initialise any obvious fields.

(Note: based on your declarations, it should be possible to use and manipulate all the information through a single pointer variable.)

(a) A book, with a title, author and year of publication.

(b) A digital image. The image has a filename of unlimited length, and a two-dimensional grid of *pixels*. The image can have any (positive) width and height. Each pixel has a single colour, made up of three components — red, green and blue. Each of these is an integer.

When allocating/initialising the structure, assume there are pre-existing integer variables `w` and `h` containing the required width and height of the image, and `nameLen` contains the length of the name.

(c) A list of sporting matches, where each match records the names of two competing teams and their integer scores. All matches are played in the same year, and this year should also be recorded. There are no limits on the number of matches, or on the length of team names. By pure chance, all team names are 10 characters long.

Question 2 appears on the next page

Question 2

Consider each of the following code snippets:

(i)

```
int *a, *b, **c, **d, j;

c = (int**)malloc(3 * sizeof(int*));
c[0] = (int*)malloc(5 * sizeof(int));
c[1] = *c + 1;
c[2] = *(c + 1) + 2;

a = &c[0][0];
b = c[0] + 3;
d = &b;
*(c + 2) = *d;

for(j = 0; j < 3; j++)
{
    c[j][0] = j;
    c[j][1] = 2 * j;
}
*a = (*d)[0] + b[a[1]];
```

(ii)

```
int **a, **b, j;

a = (int**)malloc(3 * sizeof(int*));
b = (int**)malloc(sizeof(int*));
*a = (int*)malloc(3 * sizeof(int));
*(a + 1) = (int*)malloc(4 * sizeof(int));
a[2] = a[1] + 2;
b[0] = &a[1][1];
a[0] = *a + 1;

for(j = 0; j < 3; j++)
{
    *(a[j]) = j;
    a[j][1] = 2 * j;
}
(**b)++;
b[0][2]++;
```

(a) For (i) and (ii), draw separate diagrams showing the arrays and all the pointer relationships created.

(b) For both (i) and (ii), show all `int` values at the end, except **j**. Indicate any that are left uninitialised.

Question 3 appears on the next page

Question 3

For this question, refer to the following declaration:

```
#define WIDTH 20
#define HEIGHT 30

typedef struct {
    double s;
    double t;
} Pair;
```

(a) Write a C function (not a whole program) called **processPairs()**. The function should:

- Import a fixed-size 2D array of **Pair**s, with **HEIGHT** rows and **WIDTH** columns.
- Calculate the sum (addition) of all the **s** fields.
- Calculate the product (multiplication) of all the **t** fields.
- Return nothing, but export the sum and product using another (single) **Pair** parameter, passed by reference.

(b) Write a C function (not a whole program) called **min2D()**. The function should:

- Import a fixed-size 2D array of real numbers, with **HEIGHT** rows and **WIDTH** columns (where **HEIGHT** and **WIDTH** are preprocessor constants).
- Also import a fixed-size 1D array of real numbers, with **HEIGHT** elements.
- For each row in the 2D array, determine the minimum value.
- Export the minimum values using the 1D array.
- Return nothing.

Question 4 appears on the next page

Question 4

Refer to the following standard C function prototypes (you may not need all of them):

```
FILE *fopen(char *path, char *mode);
int fclose(FILE *fp);
int fscanf(FILE *stream, char *format, ...);
char *fgets(char *s, int size, FILE *stream);
int fgetc(FILE *stream);
int ferror(FILE *stream);
size_t strlen(const char *s);
```

Also refer to the following declaration:

```
typedef struct {
    char search;
    char *text;
} Set;
```

(a) Write a function called `readFile()` to:

- Import a filename as a parameter.
- Read the contents of the file into a dynamically-allocated array of `Set` structs. The file is formatted as follows:
  - The first line contains two integers: the number of subsequent lines, and the maximum text length (see below). These values have no fixed limits.
  - Each subsequent line consists of a single non-space character, followed by a space, followed by one or more characters of free-form text. The length of the free-form text will be, at most, the maximum given on the first line.

  For example:

  ```
  4 8
  x abc
  e elephant
  3 1a2b3c
  % !@  #$%
  ```

- Return the array of `Set`s, and export the array length via a parameter passed by reference.
- 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 4 continues on the next page

(b) Write a function called **`countChars()`** to:

- Import an array of **`Set`** structs, and the array length; and return nothing.
- For each **`Set`**:
  - Count the number of occurrences of the **`search`** character within the **`word`** string. (Count exact matches only. Uppercase and lowercase letters are different.)
  - Calculate the proportion (i.e. a real number between 0 and 1) of the characters in `word` that match **`search`**.
  - Output the count and proportion on a single line, separated by a space. The proportion should be expressed with 3 decimal places.

  For example:

```
0 0.000
2 0.250
1 0.167
1 0.143
```

  (This is the expected output given the example input file shown previously.)
- Return nothing.

(c) Write a **`main()`** function to:

- Take one or more filenames on the command-line.
- For each filename, read the file with **`readFile()`** and, as appropriate, count matching characters with **`countChars()`**.
- Clean up any allocated memory.
- Output any appropriate error messages.

Question 5 appears on the next page

Question 5

(a) Given the following declarations, write a function called **`listAvg()`** to determine the mean of all real-numbers stored in the list.

```
#define ARRAYSIZE 30

typedef struct Part {
    struct Part *next;
    double array[ARRAYSIZE];
} Part;

typedef struct {
    Part *first;
} List;
```

Your function should take a single pointer to `List` and return a `double`. There should be no input or output. If the list is empty, your function should return zero. Assume the list has already been populated (i.e. filled-in/initialised).

(b) Consider the following declarations:

```
typedef struct Node {
    struct Node *next;
    char *text;
} Node;

typedef struct {
    Node *start;
    Node *end;
} List;
```

Write a function called **`listCase()`** to determine (1) the total number of lowercase characters, and (2) the total number of uppercase characters in the text stored in the list. Do not break this down per node; rather, count all lowercase and uppercase characters across all nodes.

Your function should return nothing, and take three parameters:

- A pointer to `List`.
- Two pointers to integers, called `lower` and `upper`, to export the results.

There should be no input or output, and your function should *not* modify the list.

End of Mock Test 2B