Curtin University — Department of Computing

Introduction to Programming Environments 152 / 502
(Index 10163 / 10225)

Semester 2, 2011

Mock Exam

Instructions

You should attempt this mock exam **by yourself** under **test conditions**. Answers will not be provided unless you've made a realistic attempt yourself.

Note: the *real* exam will be **closed book**, and you will be given **120 minutes**.

Question 1 appears on the next page

Question 1

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

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

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

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

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

Question 2 appears on the next page

Question 2

(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 3

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 4 appears on the next page

Question 4

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.

```
1  void searchAndReplace(char* text) {
2      char search[MAX_LENGTH + 1];
3      char replacement[MAX_LENGTH + 1];
4      int found, location = 0, len;
5      int totalLength = strlen(text);
6
7      /* Ask user for the term to search for and its replacement.
8         (The user can also cancel the operation here.)  */
9      if(readSRInput(search, replacement, MAX_LENGTH)) {
10         len = strlen(search);
11
12         /* Loop until we can't find any more matches, or we
13            reach the end of the text. */
14         do {
15             /* Find the next match. */
16             found = findNext(text, search, &location);
17             if(found) {
18                 printf("Found %s at %d\n", search, location);
19
20                 /* Ask the user whether to replace. */
21                 if(askToReplace(text, location, len)) {
22                     /* Perform the replacement. */
23                     replace(text, location, len, replacement);
24                     location += strlen(replacement);
25                 }
26                 else {
27                     location += len;
28                 }
29             }
30         }
31         while(found && location < totalLength);
32     }
33 }
```

Question 4 continues on the next page

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

For each situation below:

- State your initial hypothesis, and justify it. What possible fault/defect (in the code *not* shown) might explain the observations?
- Explain how you would use debugger features to confirm or refute your hypothesis. In particular, where would you set **breakpoints**, and which variables would you monitor?

Your hypothesis should be plausible, and your explanation should follow logically. 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 5 appears on the next page

Question 5

For all parts of this question, ensure that your C code conforms to the characteristics emphasised in the lectures and practical sessions.

(a) Design suitable structures to represent each of the following sets of information. Implement your design in C using `typedef` declarations (as they would appear 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, described by:

- The number of "core" member companies.
- The number of "associate" member companies.
- An array of core member companies.
- An array of associate member companies.

(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).
- Returns `NULL` instead if any error occurs, and outputs an appropriate error message.

(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.

(e) Write the appropriate function prototype declarations (as they would appear in a header file) for the functions from parts (b) and (c).

—— End of Mock Exam ——