Curtin University — Department of Computing

UNIX and C Programming 120 / 500
(Index 10163 / 10225)

Semester 2, 2012

Mock Test 1

Instructions

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

Note: the *real* test will be **closed book**, and you will be given **90 minutes**.

Question 1

Briefly describe each of the following declarations:

(a) `double area(double width, double height);`

(b) `int **h = NULL;`

(c) `float *f(float** x, void* y);`

(d) `void *(*coagulation)(void*);`

(e) `char **(*coagulate(void*))();`

(f) `#define SEQ(a,b,c) ((a) < (b) && (b) < (c));`

(g) `typedef void (*Spandex)(double, int);`

Question 2

(a) What is the address-of operator, and how do you use it?

(b) What is the purpose of a **rule** in a makefile?

(c) When declaring a #define macro (with parameters `x`, `y` and `z`), where should you place brackets and why?

(d) What does `NULL` mean?

(e) What is the difference between `if(x == 1)` and `#if x == 1`?

Question 3

Write a C function (*not* a whole program) called `readValidInt()`, taking no parameters and returning an `int`.

Each time your function is called, it should read one integer from the user. If the number is zero or positive (which we'll define to be "valid" for our purposes here), it should be returned.

If the number is "invalid" (i.e. negative), the last valid number supplied to the function (on a previous call) should be returned instead. If the function has not yet received a valid number on any of its calls so far, it should return -1.

Question 4

Say you have the following declarations:

```
float e = 2.0;
float f = 0.5;
float *g = NULL;
float *h = NULL;
float *i = NULL;
float **v = NULL;
float **w = NULL;
```

Based on each of the following code snippets:

- draw a diagram showing which variables point to which other variables (in the end)
- state the resulting values of `e` and `f`

(a)
```
h = &e;
v = &h;
w = v;
i = *w;
v = &g;
g = &f;
*v = h;
**w = **v * **v;
```

(b)
```
int j;
v = &g;
w = &h;
*v = &e;
*w = &f;
i = *v;
for(j = 0; j < **w; j++) {
    *i = -*i;
}
**w = -**v;
```

Question 5

Each of the following C functions contains an error. Describe what is wrong. (You don't need to show how to fix it, but you can if it will assist your explanation.)

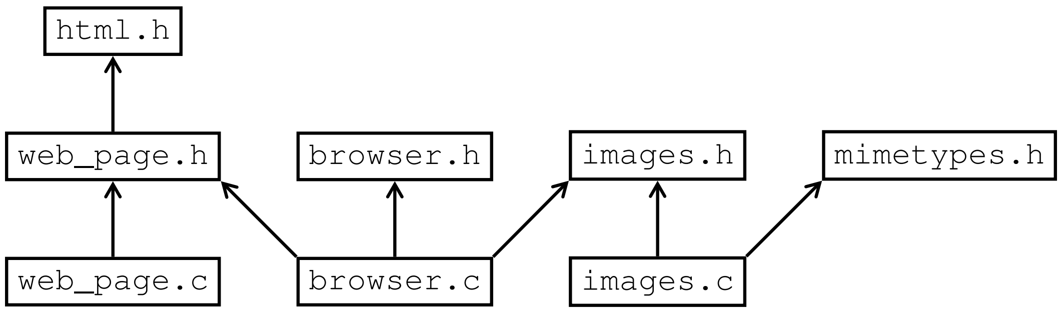
(a)
```
static double counter(double *increment)
{
    /* Maintains a running counter, which is incremented
       and returned whenever the function is called. */
    double counter = 0;
    counter = counter + *increment;
    return counter;
}
```

(b)
```
int square(int n) {
    int* x;
    *x = n * n;
    return *x;
}
```

(c)
```
void readVals(int x, int y) {
    printf("Enter first integer: ");
    scanf("%d", &x);
    printf("Enter second integer: ");
    scanf("%d", &y);
}
```

Question 6

The following diagram represents the `#include` relationships between the files making up a simple web browser.



Produce a suitable makefile for this project. Make good use of makefile variables, and include a suitable "clean" rule. The executable file should be named `browser`.

Question 7

NASA is designing a new space probe to search for life on other planets. The probe will be sent to a given planet. It will orbit the planet, map the surface and take sensor readings. The probe will look for promising locations on the surface for further exploration.

Your job is to design the software to analyse the probe's measurements. The probe will divide the planetary surface into a rectangular grid (excluding the polar regions). The non-negative grid coordinates x and y will identify each grid square. The probe will take one set of readings in each grid square. Specifically, it will record:

- temperature (°C),
- wind speed (km/h),
- oxygen concentration (%), and
- volcanic activity (yes/no).

For each set of readings (i.e. for each grid square), your software must compute a "life rating". For the planet/grid as a whole, it must also compute an "exploration risk" (an indicator of the danger faced by further, surface exploration).

If there is *no* volcanic activity in a given grid square, the life rating is as follows:

$$\text{life rating} = \frac{\text{temperature} \times \text{oxygen concentration}}{\text{wind speed}}$$

If there is volcanic activity, life rating is calculated the same way, but then divided by 3.5.

For each grid square containing volcanic activity, exploration risk goes up by 1.

Write a C function (not a whole program) to do the following:

1. Retrieve the sensor readings from the probe, using the `getReadings()` function.
2. Calculate the life rating and exploration risk, as described above.
3. Report each life rating by calling another function (through a given function pointer).
4. Report the exploration risk (via an `int*` parameter).

The `getReadings()` function is declared as follows:

```
void getReadings(int x, int y, double* temp, double* windSpeed,
                 double* oxygen, int* volcanism);
```

Your function should return `void` and take four parameters:

- `width` — an `int`, the width of the grid ($0 \leq \texttt{x} < \texttt{width}$).
- `height` — an `int`, the height of the grid ($0 \leq \texttt{y} < \texttt{height}$).
- `explorationRisk` — a pointer to an `int`.
- `reportFunc` — a pointer to a function used to report the life rating. The function takes two `int`s (grid coordinates) and a `double` (the life rating), and returns `void`.

Feel free to abbreviate any of these names, as long as they are still clear. Ensure your code conforms to the standards emphasised in the lectures and tutorials.

Question 8 appears on the next page

Question 8

A private investigator has asked you to help develop software to analyse fingerprint data. The software will have access to the left and right index fingerprints for a group of people, and also a collection of unidentified prints. The investigator wishes to know how many of the as-yet unidentified prints "possibly" or "definitely" belong to known individuals.

Each person is identified by an integer ID, ranging from zero to the number of people minus one. Each unidentified print is also identified by an integer ID, in the same fashion.

The left and right fingerprint data for each person is stored in a highly-reduced/compressed form — a "template". Each template takes only 64 bits to store.

Write a C function (*not* a whole program) called `countMatches` to do the following:

1. Retrieve each person's template fingerprints using the `getTemplates` function, which is declared as follows:

   ```
   void getTemplates(int person, long long *left,
                                 long long *right);
   ```

   If `getTemplates()` places -1 in both `*left` and `*right`, no template fingerprint exists for that person. Such people cannot be tested, of course. For efficiency purposes, your function should skip over them.

2. If template fingerprints *have* been recorded for a given person, test them against each unidentified fingerprint. Your function will be supplied with a callback (function pointer) parameter for this purpose.

3. Report the total number of "possible" and "definite" fingerprint matches, via two `int*` (pass-by-reference) parameters.

Your function should return `void`, and take five (5) parameters:

- `nPeople` — an `int`, the number of people in the database;
- `nPrints` — an `int`, the number of unidentified fingerprints;
- `nPossible` — a pointer to an `int`, the number of "possible" fingerprint matches;
- `nDefinite` — a pointer to an `int`, the number of "definite" fingerprint matches; and
- `check` — a pointer to a function used to check fingerprint templates against unidentified prints.

The callback function takes an `int` — the ID of the unidentified print — and two `long longs` — the left and right templates. It returns one of three `int` values: 0, indicating no match; 1, indicating a possible match; or 2, indicating a definite match.

Do not attempt to write the `getTemplates` function or the callback function yourself. (These have already been implemented.)

Feel free to abbreviate any of these names, as long as they are still clear. Ensure your code conforms to the standards emphasised in the lectures and tutorials.

—— End of Mock Test 1 ——