Curtin University — Department of Computing

UNIX and C Programming 120 (10163)

Mock Test 1B

Practice for Test 1

Real test weight: 15% of the unit mark.

Attempt this mock test in preparation for Test 1. 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: 28th August, 2015

Question 1 (8 marks)

Explain the meaning of the following:

(a)
```
static int f(void) {
    return 5;
}
```

(b)
```
int f;
int* g = &f;
*g = 42;
```

(c)
```
double (*p)(double,int);
```

(d)
```
void* f(int(*)(int));
```

(e)
```
typedef unsigned int (*Q)(void);
```

(f)
```
int i = 42;
#ifndef A
    i += 1;
#endif
```

(g)
```
#define M(x) ((int*)(x))
```

Question 2 appears on the next page

Question 2 (10 marks)

Consider the following code:

```
float **white, **blue;
float *purple, *yellow, *orange;
float green = 10.0, red = 20.0;

purple = &red;
orange = &green;
yellow = purple;
blue = &yellow;
white = &orange;
purple = *white;
*white = yellow;
*purple = *yellow + **blue;
**white += *orange;
```

(a) Draw a diagram showing all pointer relationships created. (Follow the approach outlined by your tutor.)

(b) State the resulting values of all non-pointer variables.

Question 3 (12 marks)

Describe the effect of the following functions, using examples as needed:

(a)
```c
void f(int* x, int* y)
{
    static int z = 0;
    if(*x == *y)
    {
        z++;
        *x = z;
    }
    else
    {
        z--;
        *y = z;
    }
}
```

(b)
```c
void g(int (*h)(int))
{
    int z, i = 0;
    do
    {
        z = (*h)(i);
        i++;
    }
    while(i <= z);
}
```

(c)
```c
double k(void* m, void* n)
{
    if(*(int*)m == 42)
    {
        printf("%f\n", *(float*)n);
    }

    return *(double*)m * *(double*)n;
}
```

Question 4 appears on the next page

Question 4 (10 marks)

Consider a C program made up of the following files:

window.c

```c
#include <stdio.h>
#include "window.h"

void displayWindow(...) {...}
```

window.h

```c
void displayWindow(...);
```

audiofile.c

```c
#include <stdlib.h>
#include "audiofile.h"

void* load(...) {...}
void play(...)  {...}
void pause(...) {...}
void stop(...)  {...}
```

audiofile.h

```c
#include "formats.h"

void* load(...);
void play(...);
void pause(...);
void stop(...);
```

player.c

```c
#include <stdlib.h>
#include <stdio.h>
#include "window.h"
#include "audiofile.h"

int main(void) {...}
```

formats.h

```c
#define FORMATS "mp3,ogg,wav,flac"
```

(a) What files would serve as targets in a makefile, and why?

(b) What are the dependencies for each target, and why?

(c) If the file `formats.h` changes, which files would need to be recompiled, and how does make figure this out?

Question 5 appears on the next page

Question 5 (40 marks)

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, double* left, double* 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 `double`s — 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.)

End of Mock Test 1B