CS240 Midterm Answers 07/10/2014

P1(a) 12 pts

Calling by reference/address: for scanf() to store the integer value read from stdin in the caller's memory location x, it needs to be provided with the address (or reference) of x.
4 pts

A segmentation fault is likely. scanf() expects a pointer and will use indirection operator * to access the caller's address of x. However, since the value of x was passed, the value, viewed as an address, will likely be nonsensical (i.e., fall outside the running app's address space).
4 pts

Since printf() only prints the value passed to stdout, it suffices for the caller to pass the value of x.
4 pts

P1(b) 12 pts

The C preprocessor scans C code for its primitives that start with # and contain keywords such as include and define. Depending on the semantics of the keyword, it carries out the requested operations which modifies the original C code. The modified C code, stripped of all C preprocessor primitives, is then input to the C compiler.
6 pts

#include reads in a file which has the effect of increasing the size of the original C code. #define replaces all instances of the symbol defined with its actual value (or expression).
6 pts

P1(c) 12 pts

If y is shared by many functions of an app, then making it global will make it easier for the functions to access (i.e., read, write) it. Conversely, if y is only (or mainly) used by some function gum(), then making it local to fun() allows other functions to reuse the variable name y locally and helps prevent other functions from accidentally gaining access to fun()'s y or corrupting its value.
4 pts

myfunc() must contain the declaration "extern float y;".
4 pts

No, unless myfunc() calls otherfunc() and passes x by value (allows read only) or reference (allows read and write).
4 pts

P2(a) 12 pts

The value of x is 10.
4 pts

The declaration defines y as an integer pointer, i.e., y contains an address (or reference) to a location that then contains an integer value. The statement "y=&x;" stores the address of integer variable x into y, and "*y=10;" stores the value 10 into the location x.

8 pts

P2(b) 12 pts

The running app will likely terminate due to a segmentation fault.
4 pts

r is declared as a pointer of float, i.e., contains an address to a location that contains a value of type float. The declaration allocates the space for r (4 bytes in a 32-bit architecture) but the address contained in r has not been specified in the code. Hence it's likely to contain an address that does not belong to the running app (i.e., an address that it is not permitted to access). When the hardware and operating system detect such a memory access attempt, the offending app is terminated.
8 pts

P2(c) 12 pts

Yes, s[3] needs to be terminated with '\0' before passing to printf() to be output as a string.
3 pts

s[0] contains 'g' and s[1] contains 'o'
3 pts

u is declared as a pointer that references char. "u=s;" stores in u the content of s. But s being declared as a 3-element array of type char is also a pointer to a location that contains a value of type char. Thus "u=s;" stores in u the address of the same location that s holds. Therefore "*u='g';" stores in the location pointed to by u the character 'g', and "*(u+1)='o';" stores in the location adjacent to the location pointed to by u the character 'o'. This is equivalent to writing "s[0]='g';" and "s[1]='o';".
6 pts

P3(a) 14 pts

```
// Prep z so that its last two bits become the first two bits.
// This is done by right shifting by 30.
z >> 30
```
4 pts

```
// Prep a mask whose bits are 0 everywhere except the first two bits.
~(~0 << 2)
```
4 pts

```
// Finally, AND the two.
(z >> 30) & ~(~0 << 2)
```
4 pts

The above may be printed as an integer which will be 0, 1, 2 or 3. They, respectively, mean 00, 01, 10, 11 as the bit values of the two most significant bits of z.
2 pts

P3(b) 14 pts

```
double *a[10];
```
a is a 10-element array where each component is a pointer to double.

4 pts

unsigned int **b;
b contains an address to a location which, in turn, contains an address to a location. This last location contains a value of type unsigned int.
4 pts

float *myfunc(int, int *);
myfunc() takes two arguments: the first of type int and the second a pointer to int. myfunc() returns a pointer to float.
3 pts

char (* yourfunc)(void);
yourfunc() does not have any arguments. yourfunc() is a function pointer, i.e., yourfunc() contains the address where another function is located.
3 pts

Bonus

Suppose we have myfunc() { static int i; } which defines i as a static integer within the scope of myfunc(). This can be useful if the programmer wants the variable name i to be reused by other functions in an app without conflicting with myfunc()'s i..
5 pts

The static way of defining a global is the same in that it is persistent as default globals are. That is, a local variable to a function only exists when a function is invoked and ceases to exist when the function returns. Globals, including static globals, are allocated memory that remains valid until an entire app terminates. It is different than default globals in that the static variable cannot be accessed outside the scope of the function where it is defined.
5 pts