CS 451/551 Quiz 1 Annotated Solution

Date: Feb 8, 2016 **Max Points**: 15

Important Reminder As per the course Academic Honesty Statement, cheating of any kind will minimally result in receiving an F letter grade for the entire course.

1. Which of the following declarations correctly declares `f` to be a function which returns a pointer to a `const int`?

   a) `const int (*f)();`

   b) `int *const f();`

   c) `const int *f();`

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

   e) `const int *const f();`

   Answer: (c).

   Using the precedence of the operators, (c) reads as `f` is a function returning a pointer to a `const int`. Note that (d) is syntactically invalid as grouping parentheses cannot be put around the type.

2. In the following declaration

   ```
   int ((*f)());
   ```

   which pairs of parentheses are redundant and may be removed without changing the meaning of the declaration?

   a) The first pair.

   b) The second pair.

   c) The third pair.

   d) The first and second pair.

   e) None of the pairs are redundant.

   Answer: (a).

   The declaration with the first pair of paren included would read as `f` is a pointer to a function returning an `int`. OTOH, if the declaration was changed to `int (*f)()`, it would read the same. Note that dropping either of the other two parentheses pairs would change the declaration; i.e. `int (*f())` would read as `f` is a function returning pointer to `int` and `int ((*f))` would read as `f` is a pointer to a `int`. Hence only the first pair of parentheses are redundant.

3. Which of the following macro definitions is safe to use like a function?

   a) `#define min(a, b) (a < b) ? a : b`

   b) `#define min(a, b) ((a < b) ? a : b)`

   c) `#define min(a, b) (((a) < (b)) ? (a) : (b))`

   d) `#define min(a, b) ((a) < (b)) ? (a) : (b)`

   e) None of the above.

   Answer: (e).

   Note that a function evaluates its arguments only once. None of the above macro definitions guarantees that the arguments are evaluated only once. If one was not concerned about argument evaluation, then definition (c) would be the best as it avoids any precedence surprises.

4. Which of the following will always constitute a definition of a top-level `a`?

   a) `int a;`

   b) `int a = 0;`

   c) `extern int a;`

   d) `int (a);`

   e) None of the above.

   Answer: (b).

   The recommended method of ensuring that we have a definition is to omit the storage specifier and include a initializer; (b) does that.

5. Given the following C code:

```
union {
  int i;
  char c[sizeof(int)];
} u = { 0x12345678 };
char x = u.c[1];
```

   assuming that a `int` occupies 4 bytes and is laid out in little-endian order, what will be the value of `x`?

   a) Undefined.

b) `0x12`.

c) `0x34`.

d) `0x56`.

e) `0x78`.

Answer: (d).

[During the exam it was announced that the above code occurs within a function (otherwise, a top-level initializer for `x` would not be legal (not a constant) and the answer would have been (a)).]

The little endian layout for `i` would be `0x78 0x56 0x34 0x12`. Since `i` and `c` occupy the same memory, `u.c[1]` would be `0x56`.

6. Which of the following statements is clearly **false**?

   a) When C was first defined, it did not have its current syntax for declaring functions (function prototypes).

   b) The size of a `union` will be the size of its largest field.

   c) The size of a `struct` will be greater than or equal to the sum of the sizes of its fields.

   d) C does not have a separate string type.

   e) A `typedef` defines a new type.

   Answer: (e).

   A `typedef` is merely an **alias** for an existing type, and does not define a new type. All the other statements are true.

7. Given the declarations:

```
char p1[] = "hello";
char *p2 = "hello";
```

   On a 64-bit machine, which of the following is true?

   a) `sizeof(p1) == 5 && sizeof(p2) == 8`

   b) `sizeof(p1) == 8 && sizeof(p2) == 8`

   c) `sizeof(p1) == 6 && sizeof(p2) == 8`

   d) `sizeof(p1) == 5 && sizeof(p2) == 5`

e) `sizeof(p1) == 6 && sizeof(p2) == 6`

Answer: (c)

The array `p1[]` would contain 6 chars (`'h' 'e' 'l' 'l' 'o' '\0'`); hence `sizeof(p1) == 6`. A pointer on a 64-bit machine would occupy 8-bytes (assuming `CHAR_BIT` is 8); hence `sizeof(p2) == 8`.