CS 1723, *C Pitfalls*

Standard pitfalls:

1. The equality operator is ==, but the following is legal C code:

```
if (x = y) ...
```

It assigns `y` to `x`, and the result is true unless `y` is `0`.
Similarly, the statement

```
n  == 10;
```

is legal: it just computes "`n==10`" as `1` if `n` is equal to `10` and `0` otherwise. Then the result is discarded.

2. The "`return`" statement returns from a function immediately.

3. If we want a function to return a value in a parameter, we must pass the address, as

```
scanf("%i", &n);
```

4. Functions with no parameters must still be called using empty parentheses, as

```
n = rand(); /*  n = rand; is wrong */
```

5. A semicolon after a for or while terminated the statement, as

```
for(i = 0; i < 10; i++) ;
     a[i] = 0;
```

This increments i to 10, and then attempts to set `a[10] = 0`.

6. Omitting a `break` after any but the last `case` in a `switch` statement.

7. In attempting to work with 2-dimensional arrays, the code below is legal C

```
int a[3, 3];
a[2, 1];
```

But it doesn't do what is intended, since the comma is taken as the C comma operator, and only 1-dimensional arrays result from this code.

8. Suppose we want to use a reference parameter to count the number of times a function is called. You might try this:

```
int i, j, count = 0;
i = func(j, &count);
```

And inside `func`:

```
int func(int n, int *countp)
{
    ...
    *countp++;  /* incorrect */
}
```

This is incorrect, since the value at the address `countp` is fetched and discarded, and then the address "`countp`" is incremented. The correct incrementing statement must use extra parentheses:

```
(*countp)++;
```

This increments the value at the address given by `countp`, which is what was desired. There are many other places where parentheses are needed to avoid problems with the precedence of operators. (This example is particularly confusing, since the operators `++` and `*` (dereference) have the same precedence, but `++` applies before `*` because the operators are applied right-to-left. However, the *effect* of the `++` only occurs after the value `*countp` is used.)

Related to Strings:

1. Consider the declaration

   ```
   char * c1, c2;
   ```

   This does not declare `c2` to be `char *`, but just `char`.

2. Given the declaration

   ```
   char *c1, *c2;
   ```

   one cannot compare the strings for equality using

   ```
   c1 == c2
   ```

   but instead one must use

   ```
   strcmp(c1, c2)
   ```

3. `strcmp` returns `0` for a successful compare.

4. The character `'\n'` is not the same as the string `"\n"`, and just a bare `\n` is illegal inside C code.

5. Strings must have the character `'\0'` at the end. Declarations for strings must allow room for this character, so that in copying a string `c1`, one would need to write

   ```
   c2 = (char *) malloc(strlen(c1) + 1);
   ```

   to allow enough room.

6. The standard copy sequence

   ```
   while (*c2++ = *c1++)    ;
   ```

   is exactly the same as

   ```
   strcpy(c2, c1);
   ```

   in case `c1` and `c2` are `char *` parameters, but when embedded in code the while loop leaves the pointers pointing past the proper beginning points of the strings.

7. Given an array declaration like

   ```
   char c3[20];
   ```

   `c3` gives the address of the string and behaves in some ways like a `char *` pointer. However, the array name is not a variable, so that operations on `c3` such as

   ```
   c3 = c1;
   ```

   or

   ```
   c3++
   ```

   are illegal.

8. In the segment

   ```
   sprintf(str, "%f", x);
   ```

   it is not enough to declare

   ```
   char *str;
   ```

   but storage but be allocated for the actual string (enough for the `sprintf` operation and for the `'\0'` at the end).
   Similarly, `strcpy(str, "Harry");` would be wrong.

9. Suppose one wants to return a string as a reference parameter from a C function. A Pascal programmer, thinking of strings as pointers and of reference parameters as pointers also, might think that just a `char *` would work. However, an extra level of indirection is needed.