Volatile

For example...

```
void delay(int count)
{
      int i;
      for (i = 0; i<count; i++) ;
}
```

In the above function without a volatile the end result is...

1. i = count;
2. i is never returned
3. so the whole routine without impacting any variables can be optimized out as a return.

Make it a volatile and you are saying all accesses to i (regardless of need must occur). Hence delay.

```
/* corrected delay */

void delay(int count)
{
      volatile int i;
      for (i = 0; i<count; i++) ;
}
```

Initialized Data

```
int globalfunction(int temp)
{
     static int staticvar=0; /* only available inside this function */
     temp = staticvar++;
     return temp;
}
```

Why this is not good coding…

1. About 2/3rds of all programmer will at initial glance think the code will return a 1 or 0. You should never write code that a very large percentage of programmers will not understand. I have seen this as a fact 7 years of teaching, informal surveys and in prior companies I wrote and reviewed tests for software candidates interviewing for jobs.
2. staticvar gets initialized in one of two ways.
    a. That location gets preset to a value of zero when the code gets downloaded into RAM. On some products if you reset you don't re-download memory you just jump to the startup code. Then that staticvars does not start over at zero. This is why some products the power cycle fixes things the reset does not (we have all had seen those issues). You can argue whether this is a systems problem but, this saves the download time which can be significant and that time could cause your product to violate specifications and even if it is a system issue software can fix it.
    b. C creates of function initVars or similar that is expected to be called prior to the code getting started (this happens in the main routine on PCs) but what happens if you have multiple mains (tasks) like an RTOS happens or none because depending on hardware you jump to different start up routines. So all those possible startup routines would need to call the initVars prior to you C code starting up.

3. This same issue occurs with all static variables that have initialize at that definition as well as globals. However if you see comment 1 then you know why this is worse for local statics initialized inside a function.
4. Since I have multiple was the code will not be understood or not work correctly. Why not do it correctly.

```
/* corrected code */

static int staticvar;
/* this function gets called in your start up code */

int initfunc(int temp)
{
    staticvar=0;
}

/* call initfunc prior to this */
int globalfunction(int temp)
{
    temp = staticvar++;
    return temp;
}
```