Assertions and Defensive Programming

C has support for assertions:

```
#include <assert.h>
assert( logical_expression );
```

At run-time, every assertion is checked by evaluating the `logical_expression` as a Boolean, and then - if it is false:

- An error message is sent to `stderr` (the console) stating the source filename, current function, and line number where the assertion failed
- Then Program execution stops immediately

Use assertions a lot

The most obvious use of assertions is in preconditions: every function that has a precondition should also - if it's feasible to check the precondition - assert that the precondition is true. For Example:

```
// divmod: performs integer division and remainder calculations
// usage: int d, m; divmod( A, B, &d, &m );
// This will set d to int(A/B), m to A mod B
// pre-condition: B must not be zero.

void divmod( int a, int b, int *dp, int *mp ) {
        assert( b != 0 );
        *dp = a/b;
        *mp = a%b;
}
```

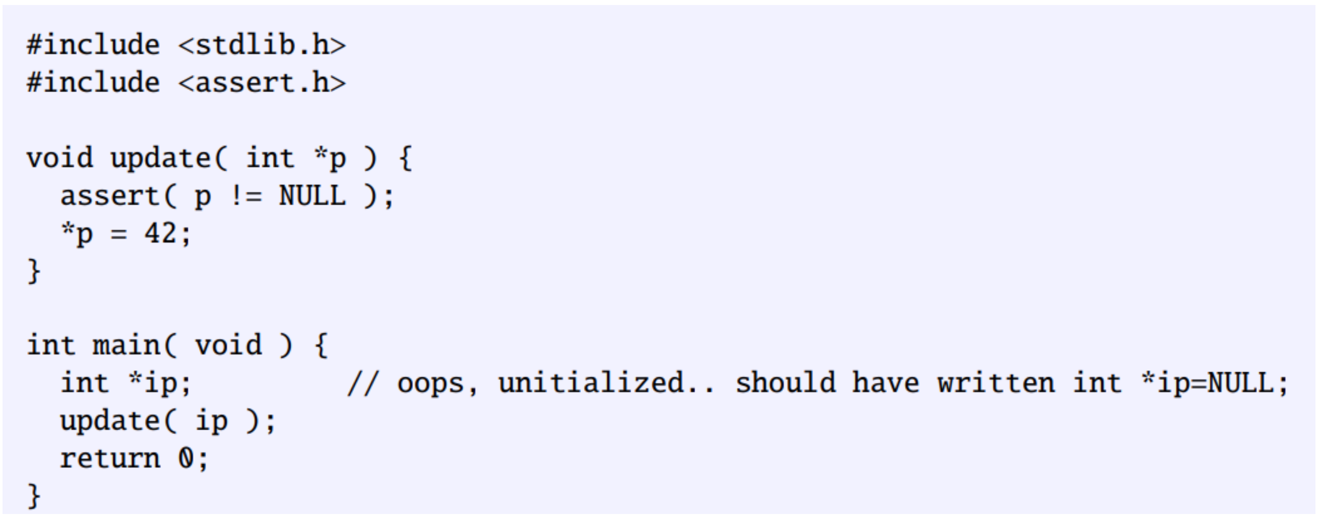
You can also use `assert` to check loop invariants and post-conditions

However, you should only assert logical truths, so don't use `assert` to handle a normal error - something that happens if the user is careless:

```
int main( int argc, char **argv ) {
        assert( argc>2 );
        ...
}
```

You can nullify any pointer by setting it to NULL - this is defined in `stdlib.h`, usually as 0

Checking whether a pointer is NULL is a sensible check, but only if it was actually set to NULL in the first place, unlike here:



```
#include <stdlib.h>
#include <assert.h>

void update( int *p ) {
  assert( p != NULL );
  *p = 42;
}

int main( void ) {
  int *ip;         // oops, unitialized.. should have written int *ip=NULL;
  update( ip );
  return 0;
}
```

Fortunately, `gcc` notices simple cases and gives a warning