Error Handling

Error handling (our first attempt)

```
char *read_file( char * path, int *size ) {
        FILE *in = fopen(path, "rb");
        if( in == NULL ) return NULL;
        int res = fseek(in, 0, SEEK_END);
        if( res == -1 ) { fclose(in); return NULL; }
        long file_size = ftell(in);
        if( file_size == -1 ) { fclose(in); return NULL; }
        res = fseek(in, 0, SEEK_SET);
        if( res == -1 ) { fclose(in); return NULL; }
        char *data = malloc(file_size);
        if( data == NULL ) { fclose(in); return NULL; }
        res = fread(data, 1, file_size, in);
        if( res < file_size && ferror(in) ) { free(data);
fclose(in); return NULL; }
        res = fclose(in); in = NULL;
        if( res != 0 ) { free(data); return NULL; }
        return data;
}
```

This is ugly

We see that our 3 macros use recovery actions that are slight variants of each other. The general case is:

```
free(data);
fclose(in);
return NULL;
```

But sometimes particular steps are left out

We want to tweak things so that every failure does the exact same sequence of recovery actions, so that we only need one macro

One problem is that the `data` variable is only declared in the second half of the code. But suppose we move our `data` variable declaration up to the top of `read_file()`, initializing it to NULL

We are allowed to `free(data)` even if `data==NULL`, so now we can always `free(data)` - even before we have `malloc()`ed it

That leaves the `fclose(in)` call. It's not safe to call that when `in==NULL` but we could protect it with `if( in != NULL ) fclose(in)`

So we end up with:

```
#define TESTFAIL(c) {if(c){free(data); if(in!=NULL)fclose(in);
return NULL;}}
char *read_file( char * path, int *size ) {
        char *data = NULL;
        FILE *in = fopen(path, "rb"); TESTFAIL( in == NULL );
        int res = fseek(in, 0, SEEK_END); TESTFAIL( res == -1 );
        long file_size = ftell(in); TESTFAIL( file_size == -1 );
        res = fseek(in, 0, SEEK_SET); TESTFAIL( res == -1 );
        data = malloc(file_size); TESTFAIL( data == NULL );
        res = fread(data, 1, file_size, in); TESTFAIL( res <
file_size && ferror(in) );
        res = fclose(in); in = NULL; TESTFAIL( res != 0 );
        return data;
}
```

Much cleaner

Summary

C does not have exceptions

Start by checking the error codes, and deciding how to handle them in your situation. If you can then identify patterns in the checks and recovery actions, macros can help in tidying away most of the nasty error handling and exposing the main core of your code