

Re: Two macros for resource management

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2004-08/0045.html

From: Stefan Ram (ram_at_zedat.fu-berlin.de)

Date: 08/01/04

Date: 1 Aug 2004 14:11:49 GMT

ram@zedat.fu-berlin.de (Stefan Ram) writes:

```
> TRY( source = fopen( "source", "r" ))
> PUSH( fclose( source ))
> TRY( target = fopen( "target", "w" ))
> PUSH( fclose( target ))
> TRY( buffer = malloc( 1024 ))
> PUSH( free( buffer ))
> result = buffcopy( target, source, buffer );
> return result; }
```

In the above posting, I raised the question how to define "TRY" and "PUSH" and I now will reply to some Followup-postings and give my own solution.

Richard Bos writes:

```
>This is not a good idea at all. Every single maintenance
>programmer who reviews this code will put semicolons after
>these "statements", thus breaking your macros.
```

This should be taken into account, when deciding whether to use this approach.

These two macros are of limited use for production code. Their main goal is to show a way to look a C code and resource management that might not be obvious.

Peter Nilsson writes:

```
> int stack[3], *sptr = &stack[sizeof stack / sizeof *stack];
>
> #define TRY(alloc) \
> for (*--sptr = 0; \
> *sptr != 0 \
> || (*sptr == 0 && (alloc) != 0) \
> || (*sptr == 0 && (sptr++, 0)) \
> ; )
>
> #define PUSH(dealloc) \
> if(++sptr == 2) { (dealloc); sptr++; break; } else
```

comp.lang.c: Re: Two macros for resource management

This is indeed a working solution, which differs from mine, although both use a for-loop.

> *If there's more than one statement to be performed, it needs to be enclosed in a block.*

Yes.

> *I'd prefer a solution where TRY and PUSH statements are ; terminated. You also lose control over fclose return values. Regaining control using these macros would be more trouble than it's worth. Even if it wasn't, I still wouldn't use this. ; -)*

The approach taken is indeed of limited flexibility.

I do not believe that I would use this approach often in production code, but still like the idea to abstract from the details of resource management.

**

Now, it will be shown how to arrive at my own solution. The first program is doing what is required (except for possible inspection of the results of "fclose"), but without those two macros.

```
-----  
| Warning: When the following programs are executed, they |  
| might overwrite (delete) an existing file named "target". |  
-----
```

```
/* < filename = [twomac0.c] > */  
  
#include <stdlib.h> /* malloc, free, EXIT_FAILURE */  
#include <stdio.h> /* FILE, fopen, fclose */  
#include "example.h" /* buffcopy */  
  
int main( void )  
{ int result = EXIT_FAILURE;  
  FILE * source;  
  FILE * target;  
  char * buffer;  
  if( source = fopen( "source", "r" ) )  
  { if( target = fopen( "target", "w" ) )  
    { if( buffer = malloc( 1024 ) )  
      { result = buffcopy( target, source, buffer );  
        free( buffer ); }  
      fclose( target ); }  
    fclose( source ); }  
  return result; }
```

comp.lang.c: Re: Two macros for resource management

The code is rewritten to use a while-statement:

```
/* < filename = [twomac1.c] > */

#include <stdlib.h> /* malloc, free, EXIT_FAILURE */
#include <stdio.h> /* FILE, fopen, fclose */
#include "example.h" /* buffcopy */

int main( void )
{ int result = EXIT_FAILURE;
  FILE * source;
  FILE * target;
  char * buffer;
  int try = 1;
  { while( try &&( source = fopen( "source", "r" )))
    { while( try &&( target = fopen( "target", "w" )))
      { while( try &&( buffer = malloc( 1024 )))
        { result = buffcopy( target, source, buffer );
          try = 0; free( buffer ); }
        try = 0; fclose( target ); }
      try = 0; fclose( source ); }}
  return result; }
```

The variable "try" was used to have each loop be executed at most once. Thus, it helps "emulating" the if-statements using the while-statements. Why was the while-loop introduced above at all? Because the code now can be rewritten to use for-statements:

```
/* < filename = [twomac2.c] > */

#include <stdlib.h> /* malloc, free, EXIT_FAILURE */
#include <stdio.h> /* FILE, fopen, fclose */
#include "example.h" /* buffcopy */

int main( void )
{ int result = EXIT_FAILURE;
  FILE * source;
  FILE * target;
  char * buffer;
  for
  ( int try = 1;
    try &&( source = fopen( "source", "r" ));
    try = 0, fclose( source ) )
  for
  ( try = 1;
    try &&( target = fopen( "target", "w" ));
    try = 0, fclose( target ) )
  for
  ( try = 1;
    try &&( buffer = malloc( 1024 ));
```

comp.lang.c: Re: Two macros for resource management

```
try = 0, free( buffer ))
result = buffcopy( target, source, buffer );
return result; }
```

Now, one can see how the "magic" works. If one does not like macros, the for-statements can be used as they are given above to be able to write close-statements near to their corresponding open statement. Otherwise, they might be packed into macros:

```
/* < filename = [twomac.c] > */

#include <stdlib.h> /* malloc, free, EXIT_FAILURE */
#include <stdio.h> /* FILE, fopen, fclose */
#include "example.h" /* buffcopy */

#define TRY(x) for(int try=1;(try&&(x));
#define PUSH(y) ((try=0),(y)))

int main( void )
{ int result = EXIT_FAILURE;
  FILE * source;
  FILE * target;
  char * buffer;
  TRY( source = fopen( "source", "r" ))
  PUSH( fclose( source ))
  TRY( target = fopen( "target", "w" ))
  PUSH( fclose( target ))
  TRY( buffer = malloc( 1024 ))
  PUSH( free( buffer ))
  result = buffcopy( target, source, buffer );
  return result; }
```

The next program features a debug-version of the code. It can be executed as it is, because it does not contain a reference to the undefined function "buffcopy".

The comparison in the function "mymalloc" might be modified to make the function "mymalloc" return 0 (by changing the value "2" to "1", "0" or "-1"). This might be used to test all possible branches of control flow and to observe that the function "myfree" only is called for those calls of the function "mymalloc", which did not return 0; and that the function "printf" only is called if all attempts did succeed.

```
/* < filename = [twomacd.c] > */

#include <stdlib.h> /* EXIT_SUCCESS, EXIT_FAILURE */
#include <stdio.h> /* fprintf, printf */
```

comp.lang.c: Re: Two macros for resource management

```
#define TRY(x) for( int try=1; (try&&(x));
#define PUSH(y) ((try=0),(y))

void *
mymalloc
( size_t const s )
{
    static int i = 0;
    void * const result = i > 2 ? 0 : malloc( s );
    fprintf( stderr, "%p = mymalloc( %zu );\n", result, s );
    ++i;
    return result; }

void
myfree
( char * const p )
{
    fprintf( stderr, "free( %p );\n", p );
    free( p ); }

int
main
( void )
{
    int result = EXIT_FAILURE;
    char *a;
    int *b;
    double *c;

    TRY( a = mymalloc( 2 * sizeof( *a )))
    PUSH( myfree( a ))
    TRY( b = mymalloc( 3 * sizeof( *b )))
    PUSH( myfree(( char * )b ))
    TRY( c = mymalloc( 4 * sizeof( *c )))
    PUSH( myfree(( char * )c ))

    { printf( "%p %p %p\n", ( void * )a, ( void * )b, ( void * )c );
      result = EXIT_SUCCESS; }

    return result; }
```