```
/*============================================================================*/
/*   MODULE                       STACK.H                                     */
/*============================================================================*/
/*   FUNCTION      Macros for creating stacks of simple objects.
 *
 *   SYSTEM        Standard (ANSI/ISO) C.
 *                 Tested on PC/MS DOS 5.0 (MSC 600A).
 *
 *   PROGRAMMER    Allan Dystrup.
 *
 *   COPYRIGHT     (c) Allan Dystrup, 1991
 *
 *   VERSION       $Header: d:/cwk/kf/h/RCS/stack.h 1.1 92/10/25 17:19:21
 *                 Allan_Dystrup Exp Locker: Allan_Dystrup $
 *                 ----------------------------------------------------------
 *                 $Log:stack.h $
 *                 Revision 1.1  92/10/25  17:19:21  Allan_Dystrup
 *                 Initial revision
 *
 *============================================================================*/

#ifndef _STACK_H                    /* Make sure stack.h is included once */
#define _STACK_H                    /* Matching #endif is at end of file. */


/*----------------------------------------------------------------------------*/
/*                             Define/Reset Stack                             */
/*----------------------------------------------------------------------------*/
/* Storage class for stack : auto (default) or static */
#define stack_cls  /*auto*/

/* Create a stack of 'size' objects, each of type 'type' (simple or ptr)   */
#define stack_dcl(stack,type,size)                \
           typedef type t_##stack;                \
           stack_cls t_##stack stack[size];       \
           stack_cls t_##stack (*p_##stack) = stack + (size)

/* Reset stack to initial (empty) condition, discarding all stack elements */
#define stack_clear(stack)            \
               ( (p_##stack) = (stack + sizeof(stack)/sizeof(*stack)) )


/*----------------------------------------------------------------------------*/
/*                             Test stack                                     */
/*----------------------------------------------------------------------------*/
/* Test for stack full : Evaluates to TRUE, if the stack is full */
#define stack_full(stack)             \
             ( (p_##stack) <= stack )

/* Test for stack empty : Evaluates to TRUE, if the stack is empty */
#define stack_empty(stack)            \
             ( (p_##stack) >= (stack + sizeof(stack)/sizeof(*stack)) )

/* Get number of elements : Evaluate #elements currently on the stack */
#define stack_ele(stack)              \
             ( (sizeof(stack)/sizeof(*stack)) - (p_##stack-stack) )


/*----------------------------------------------------------------------------*/
/*                         Push/Pop without BorderTest                        */
/*----------------------------------------------------------------------------*/
/* Push an element 'x' onto the stack */
#define push_(stack, x)            ( *--p_##stack = (x) )

/* Pop top element from the stack, usage : x = pop_(stack) */
```

```
#define pop_(stack)                ( *p_##stack++ )

/* Pop multiple elements : pop 'amt' elem; evaluate to topelem before pop  */
#define popn_(stack, amt)          ( (p_##stack += amt)[-amt] )

/* Access arbitrary emelent : Evaluate to item at indicated offset from top*/
#define stack_item(stack, offset)  ( *(p_##stack + (offset)) )

/* Access stack pointer directly : Evaluate to internal name for stack ptr */
#define stack_p(stack)             p_##stack


/*-------------------------------------------------------------------------*/
/*                         Push/Pop *with* BorderTest                      */
/*-------------------------------------------------------------------------*/
#define push(stack, x)   ( stack_full(stack)           \
               ? ((t_##stack) (long)(stack_err(1)))    \
               : push_(stack,x) )

#define pop(stack)       ( stack_empty(stack)          \
               ? ((t_##stack) (long)(stack_err(0)))    \
               : pop_(stack) )

#define popn(stack, amt) ( (stack_ele(stack) < amt)    \
               ? ((t_##stack) (long)(stack_err(0)))    \
               : popn_(stack,amt) )


/*-------------------------------------------------------------------------*/
/*                         Stack Error Handling                            */
/*-------------------------------------------------------------------------*/
#define EStk0  0               /* Error codes */
#define EStk1  1

#define stack_err(o)     ( (o)      \
               ? vError(EStk0) \
               : vError(EStk1) )



#endif /* #ifdef _STACK_H */
/* End of module stack.h                                                   */
/*=========================================================================*/
```