CHAPTER 20

PORTABILITY

20.1 PRINCIPLES OF PORTABILITY

20.2 PORTING TO NON-ANSI COMPILERS

- Dealing with Function Prototypes
- Generic Pointers
- Missing Header Files

20.3 PORTING ACROSS ANSI-C COMPILERS

20.4 PORTABILITY ACROSS OPERATING SYSTEMS

20.5 MACHINE DEPENDENCIES

- Data Type Sizes
- Character Set Differences
- Sign-Related Problems
- Byte Ordering
- Byte Alignment Problems
- Pointer Problems
- Evaluation Order
- Libraries and Machines

20.6 CASE STUDY—PORTABLE I/O DEVICE HANDLING

- Immediate Input
- Cursor Control

getline.h

(Figure 20.1 – Page 503)

```
/*
 * Defining getline's prototype.
 */
#ifdef __STDC__
int getline(char *, int);
#else
int getline2();
#define getline(l,m)  getline2((char *) (l), (int) (m))
#endif
```

getline.c

(Figure 20.2 – Page 503)

```
/*
 * Defines a function to read an input line into an array.
 */
#include <stdio.h>
#include "getline.h"

#ifdef __STDC__
int getline(char *ptr, int max)
#else
int getline2(ptr, max)
  char *ptr; int max;
#endif
{
  int  c;                          /* current character */
  char *startptr = ptr;            /* pointer to start of line */
  char *endptr = ptr + max;        /* pointer to end of line */

  while ((c = getchar ()) != '\n' && c != EOF)
    if (ptr < endptr)
      *ptr++ = c;
  *ptr = '\0';                     /* terminate with null */
  return (c == EOF) ? -1 : ptr - startptr;
}
```

gens.h

(Figure 20.3 – Page 504)

```
/*
 * Definition of generic pointer type.
 */
#ifdef __STDC__
typedef void *GENPTR;                /* ANSI generic pointer */
#else
typedef unsigned char *GENPTR;       /* no generic pointer */
#endif
```

allocs.h

(Figure 20.4 – Page 505)

```
/*
 * Useful macros for dynamically allocating things.
 */
#include "gens.h"                    /* for generic pointer */

#ifdef __STDC__       /* ANSI C */

#include <stdlib.h>                  /* for malloc/calloc/free */
#include <stddef.h>                  /* for NULL */
#include <string.h>                  /* for memcpy/memset/strcpy etc...*/

#define ALLOC(n,t)       (malloc((n) * sizeof(t)))
#define ALLOC_ZERO(n,t)  (calloc(n, sizeof(t)))

#else                 /* Old-style C */

#include <stdio.h>                   /* for NULL */
extern GENPTR malloc();
extern GENPTR calloc();
extern int free();

#define ALLOC(n,t)       ((t *) (malloc((n) * sizeof(t))))
#define ALLOC_ZERO(n,t)  ((t *) (calloc(n, sizeof(t))))

#endif

#define DEALLOC(x)       ((t *) (free((GENPTR) x)))
#define COPY(np,op,n,t)  (memcpy(np,op,(n) * sizeof(t)))
#define ZERO(p,n,t)      (memset(p,(n) * sizeof(t),0))
#define STRALLOC(s)      (malloc(strlen(s) + 1))
#define STRDUP(ns,os)    ((ns = STRALLOC(os)) ? strcpy(ns,os) : NULL)

#define TABLEDUP(np,op,n,t) \
        (((np = ALLOC(n,t)) != NULL) ? COPY(np,op,n,t) : NULL)
```

sets.h

(Figure 20.5 – Page 506)

```
/*
 * Improved definitions to use "sets" of integers.
 */
#include "allocs.h"

typedef GENPTR SET;

#ifdef __STDC__            /* ANSI C */

extern SET   set_create(unsigned int n);
extern void  set_add(SET s, int e);
extern void  set_delete(SET s, int e);
extern int   set_member(SET s, int e);

#else                      /* Old-style C */

extern SET   set_create2();
extern void  set_add2();
extern void  set_delete2();
extern int   set_member2();

#define set_create(n)     set_create2((unsigned int) n)
#define set_add(s,e)      set_add2((SET) s, (int) e)
#define set_delete(s,e)   set_delete2((SET) s, (int) e)
#define set_member(s,e)   set_member2((SET) s, (int) e)

#endif

#define set_destroy(s)    (DEALLOC(s))
#define set_switch(s,t,e) (set_delete(s,e), set_add(t,e))
```

sets.c

(Figure 20.6 – Page 507)

```
/*
 * Functions to handle dynamically allocated sets.
 *   set_create - create an empty set.
 *   set_add - add an element to a set.
 *   set_delete - remove element from a set.
 *   set_member - is item in set?
 */
#include "sets.h"

#define US_BITS 16
#define WORD(elem) ((elem) / US_BITS)
#define BIT(elem)  ((elem) % US_BITS)

typedef unsigned short USHORT;

#ifdef __STDC__
SET set_create(unsigned int n)
#else
SET set_create2(n) int n;
#endif
{
  return (SET) ALLOC_ZERO(n / US_BITS + (n % US_BITS != 0), USHORT);
}
```

sets.c

(Figure 20.6 – Page 507)

```
#ifdef __STDC__
void set_add(SET s, int elem)
#else
void set_add2(s, elem) SET s; int elem;
#endif
{
  * ((USHORT *) s + WORD(elem)) |= 1 << BIT(elem);
}

#ifdef __STDC__
void set_delete(SET s, int elem)
#else
void set_delete2(s, elem) SET s; int elem;
#endif
{
  * ((USHORT *) s + WORD(elem)) &= ~(1 << BIT(elem));
}

#ifdef __STDC__
int set_member(SET s, int elem)
#else
int set_member2(s, elem) SET s; int elem;
#endif
{
  return ((unsigned) (* ((USHORT *) s + WORD(elem))) >> BIT(elem)) & 01;
}
```

usesets.c
(Figure 20.7 – Page 508)

```
/*
 * Identify duplicates in the input (more portable version).
 */
#include <stdio.h>
#include "sets.h"
#include "getline.h"

#define MAXLEN       80
#define ELEMENTS    512

int main()
{
  SET  unique = set_create(ELEMENTS);
  SET  dup    = set_create(ELEMENTS);
  int  value;                                  /* current input value */
  char line[MAXLEN + 1];                      /* current input line */

#ifdef __STDC__
  void set_print(const char *msg, SET set, int n);
#else
  void set_print();
#endif
```

usesets.c

(Figure 20.7 – Page 508)

```
  while (getline(line, MAXLEN) != -1)
    if (sscanf(line, "%i", &value) != 1)
      fprintf(stderr,"Skipping bad input item\n");
    else if (value < 0 || value >= ELEMENTS)
      fprintf(stderr,"Out of range item %i\n", value);
    else if (set_member(unique, value))
      set_switch(unique,dup,value);          /* move to dup */
    else if (!set_member(dup, value))
      set_add(unique, value);
  set_print("Unique values\n", unique, ELEMENTS);
  set_print("Duplicate values\n", dup, ELEMENTS);
  return 0;
}

#ifdef __STDC__
void set_print(const char *msg, SET set, int n)
#else
void set_print(msg, set, n) char *msg; SET set; int n;
#endif
{
  int i;

  if (msg != NULL)
    printf(msg);
  for (i = 0; i < n; i++)
    if (set_member(set, i))
      printf("%i\n", i);
}
```

power.c

(Figure 20.8 – Page 509)

```
/*
 * Using the math library function to compute exponents.
 */
#include <stdio.h>

#ifdef __STDC__
#include <math.h>                /* grab prototypes */
#define POW(x,y)    pow(x,y)
#else
double pow();                    /* supply prototypes ourselves */
#define POW(x,y)    pow((double) (x), (double) (y))
#endif

int main()
{
  printf("10^6 = %g\n",    POW(10, 6));
  printf("3.3^-2 = %g\n",  POW(3.3, -2));
  printf("-11.5^3 = %g\n", POW(-11.5,3));
  return 0;
}
```

ANSI-C COMPILER LIMITS

TYPE	LIMIT
Nesting levels of compound statements	15
Nesting levels in conditional compilation	6
Nested declarations for a basic type	6
Expressions nested by parentheses	127
Significant characters in a macro or *internal* name	31
Significant characters in any *external* name	6
External identifiers in a source file	511
Identifiers with block scope in one block	127
Macro identifiers simultaneously defined	1024
Parameters in a function definition or call	31
Parameters in a macro definition or call	31
Characters in a logical source line	509
Characters in a string literal	509
Bytes in an object	32K
Levels of nested **`#include`** files	8
Case labels in **`switch`**	255

stacks.h

(Figure 20.9 – Page 511)

```
/*
 * Definitions of stack data type and prototypes.
 */
#include "lists.h"

typedef LISTNODE *STACK;

#define stack_create()   NULL
#define stack_empty(s)   ((s) == NULL)
#define stack_top(s)     ((s)->dataptr)

#if defined(SHORTNAMES)

#define stack_push(s,i)  stpsh(s,i)
#define stack_pop(s)     stpop(s)

#endif

STACK stack_push(STACK s, void *dataptr);
STACK stack_pop(STACK s);
```

filesize.h

(Figure 20.10 – Page 512)

```c
/*
 * Prototype and necessary definition for filesize function.
 */
#include <stdio.h>

#define FILESIZE_UNKNOWN  -1L

long filesize(const char *file);
```

filesize.c

(Figure 20.11 – Page 513)

```
/*
 * Determine the size of the given file.  Returns the special
 * value FILESIZE_UNKNOWN if it can't do so.
 */
#include <stdio.h>
#include "filesize.h"

#if defined(__MSDOS__)
#include <io.h>                    /* MS-DOS specific include file */

long filesize(const char *file)
{
  FILE *fp = fopen(file, "rb");
  long len = FILESIZE_UNKNOWN;

  if (fp != NULL)
  {
    len = filelength(fileno(fp));
    fclose(fp);
  }
  return len;
}
#elif defined(unix)
#include <sys/types.h>             /* UNIX specific include files */
#include <sys/stat.h>
```

filesize.c

(Figure 20.11 – Page 513)

```
long filesize(const char *file)
{
  struct stat s;

  return (stat(file, &s) != -1) ? s.st_size : FILESIZE_UNKNOWN;
}
#else    /* default version */
long filesize(const char *file)
{
  FILE *fp = fopen(file, "rb");
  long len = FILESIZE_UNKNOWN;

  if (fp != NULL)
  {
    if (fseek(fp, 0L, 2) != -1)
      len = ftell(fp);
    fclose(fp);
  }
  return len;
}
#endif
```

testfs.c

(Figure 20.12 – Page 514)

```c
/*
 * Print size in bytes for each of its arguments.
 */
#include <stdio.h>
#include "filesize.h"

int main(int argc, char *argv[])
{
  long fs;                                /* next file's size */
  int i;                                  /* index through args */

  for (i = 1; i < argc; i++)
    if ((fs = filesize(argv[i])) == FILESIZE_UNKNOWN)
      fprintf(stderr,"Can't get size of %s\n", argv[i]);
    else
      printf("Size of %s: %li\n", argv[i], fs);
  return 0;
}
```

ourtypes.h

(Figure 20.13 – Page 516)

```
/*
 * Defining a "SMALL CouNTer" type.
 */
#include <limits.h>

#if UCHAR_MAX >= 255
  typedef unsigned char  SMALL_CNT;          /* 8-bit chars */
#else
  typedef unsigned short SMALL_CNT;          /* 7-bit chars */
#endif
```

ourtypes.c

(Figure 20.14 – Page 516)

```
/*
 * Using the "SMALL_CouNTer" type.
 */
#include <stdio.h>
#include "ourtypes.h"                          /* type definitions */

#define MAXCNT 10

int main()
{
  SMALL_CNT table[MAXCNT];                     /* array of that type */
  SMALL_CNT i;

  for (i = 0; i < MAXCNT; i++)                 /* some initial values */
    table[i] = i;
  for (i = 0; i < MAXCNT; i++)                 /* print them */
    printf("%u\n", (unsigned int) table[i]);
  return 0;
}
```

convchar.c

(Figure 20.15 – Page 517)

```
/*
 * Functions to convert digits to and from characters (ASCII ONLY).
 *   todecimal - convert char to equivalent decimal number.
 *   tochar - convert decimal number to equivalent character.
 */
#include <ctype.h>
#include <string.h>

#define OFFSET(c,s) ((const char *const) strchr(s,c) - (s))

static const char *const Digits = "0123456789";
static const char *const LowerCase = "abcdefghijklmnopqrstuvwxyz";
static const char *const UpperCase = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

int todecimal(int c, int toobig)
{
  int value = -1;                 /* simplify later decimal handling */

  if (isdigit(c))
    value = OFFSET(c, Digits);
  else if (islower(c))
    value = OFFSET(c, LowerCase) + 10;
  else if (isupper(c))
    value = OFFSET(c, UpperCase) + 10;
  if (value >= toobig)            /* make sure result is legal */
    value = -1;
  return value;
}

int tochar(int value)
{
  return (value < 10) ? Digits[value] : UpperCase[value - 10];
}
```

baddisp.c

(Figure 20.16 – Page 518)

```
/*
 * A nonportable version of our display program.
 */
#include <stdio.h>

int main()
{
  char c;

  while ((c = getchar()) != EOF)
    putchar(c);
  return 0;
}
```

unport.c

(Figure 20.17 – Page 519)

```
/*
 * Comparing signed and unsigned values.
 */
#include <stdio.h>

int main()
{
  unsigned int i = 1U;
  long l = -1L;

  if (i > l)
    printf("As expected: 1U > -1L\n");
  else
    printf("Unexpected: 1U < -1L\n");
  if ((long) i > l)
    printf("As expected: 1L > -1L\n");
  else
    printf("Impossible: 1L < -1L\n");
  return 0;
}
```

shifts.c

(Figure 20.18 – Page 519)

```
/*
 * Computing the largest integer, correctly and incorrectly.
 */
#include <stdio.h>

#define BAD_MAXINT  ~0 >> 1                /* not portable */
#define MAXINT      (unsigned) ~0 >> 1     /* portable */

int main()
{
  printf("~0 >> 1 = %i\n", BAD_MAXINT);
  printf("(unsigned) ~0 >> 1 = %i\n", MAXINT);
  return 0;
}
```

WORD ORGANIZATION DIFFERENCES

1	0
08	$F0$
MSB	LSB

(a) On some machines the most significant byte (MSB) is the high byte.

1	0
$F0$	08
LSB	MSB

(b) On other machines, it’s the low byte.

bytetest.c

(Figure 20.20 – Page 520)

```
/*
 * Nonportable program with byte-order assumptions built in.
 */
#include <stdio.h>

union word
{
  short x;                              /* use as word */
  unsigned char bytes[2];               /* use as bytes */
};

int main()
{
  union word w;

  w.x = 0x08F0;                         /* 08 F0 */
  printf("LSB=%x, MSB=%x.\n", w.bytes[0], w.bytes[1]);
  return 0;
}
```

Sample Output – offset

```
Offset to number:       0 bytes
Offset to name:         4 bytes
Offset to phone:        44 bytes
Offset to age:          60 bytes
```

offset.c

(Figure 20.21 – Page 522)

```
#include <stdio.h>
#include <stddef.h>

#define MAXNAME 40

typedef struct employee
{
  long number;                          /* employee number */
  char name[MAXNAME];                   /* first and last name */
  char phone[13];                       /* xxx-xxx-xxxx\0 */
  int age;                              /* age */
} EMP;

#define display_offset(field, bytes) \
  printf("Offset to " #field ":  \t%li bytes\n", (long) bytes)

int main()
{
  size_t number_off = offsetof(EMP, number);
  size_t name_off   = offsetof(EMP, name);
  size_t phone_off  = offsetof(EMP, phone);
  size_t age_off    = offsetof(EMP, age);

  display_offset(number, number_off);
  display_offset(name,   name_off);
  display_offset(phone,  phone_off);
  display_offset(age,    age_off);
  return 0;
}
```

funccall.c

(Figure 20.22 – Page 523)

```
/*
 * Portable and nonportable function calls.
 */
#include <stdio.h>

int main()
{
  int func1(void), func2(void);
  int i;
  int tmp1, tmp2, tmp3;

  i = 0;
  printf("%i, %i, %i\n", ++i, ++i, ++i);          /* nonportable */

  i = 0;
  tmp1 = ++i, tmp2 = ++i, tmp3 = ++i;
  printf("%i, %i, %i\n", tmp1, tmp2, tmp3);       /* portable */

  printf("%i, %i\n", func1(), func2());           /* nonportable */

  tmp1 = func1();
  tmp2 = func2();
  printf("%i, %i\n", tmp1, tmp2);                 /* portable */
  return 0;
}

int func1(void)
{
  printf("Called func1.\n");
  return 1;
}

int func2(void)
{
  printf("Called func2.\n");
  return 2;
}
```

ichars.h

(Figure 20.23 – Page 525)

```
/*
 * Prototypes for read-immediate functions.
 */
#if defined(__TURBOC__)

#include <conio.h>

#define ichar_get()     getche()       /* built-in insta-char */
#define ichar_init()    1              /* fake function return */
#define ichar_done()    1              /* fake function return */

#elif defined(unix)

#define ichar_get()     getchar()

int ichar_init(void);
int ichar_done(void);

#else

#error Can only handle Turbo C (under MS-DOS) and UNIX

#endif
```

ichars.c

(Figure 20.24 – Page 526)

```
/*
 * Package for reading characters immediately.
 *   ichar_init - set things up for character-at-a-time I/O.
 *   ichar_done - restore things to the normal startup state.
 */
#include <stdio.h>
#include "ichars.h"

#if defined(unix)

#if defined(sys5)                   /* ATT UNIX */
#include <sys/termio.h>
#define GETTERMINFO TCGETA
#define SETTERMINFO TCSETAF

struct termio oldt, t;              /* old and new terminal status */
#else                               /* otherwise assume Berkeley UNIX */
#include <sgtty.h>
#define GETTERMINFO TIOCGETP
#define SETTERMINFO TIOCSETN
```

ichars.c

(Figure 20.24 – Page 526)

```
struct sgtty oldt, t;              /* old and new terminal status */
#endif

int ichar_init(void)
{
  setvbuf(stdin, NULL, _IONBF, NULL);
  if (ioctl(0, GETTERMINFO, &t) == -1)
    return -1;
  oldt = t;                        /* save terminal settings */

#if defined(sys5)
  t.c_lflag &= ~ICANON;            /* set up for one char read */
  t.c_cc[4] = 1;
  t.c_cc[5] = 0;
#else
  t.sg_flags |= CBREAK;            /* set up for one char read */
#endif

  return ioctl(0, SETTERMINFO, &t) != -1 ? 0 : -1;
}

int ichar_done(void)
{
  setvbuf(stdin, NULL, _IOFBF, BUFSIZ);
  return ioctl(0, SETTERMINFO, &oldt) != -1 ? 0 : -1;
}
#endif
```

yesorno.c

(Figure 20.25 – Page 527)

```
/*
 * Program using immediate input to get a yes or no answer.
 */
#include <stdio.h>
#include <ctype.h>
#include "ichars.h"

int yesorno(const char *prompt)
{
  int answer;                        /* user entered character */

  (void) ichar_init();               /* assume worked */
  for (;;)
  {
    printf(prompt);
    answer = tolower(ichar_get());
    putchar('\n');
    if (answer != 'y' && answer != 'n')
      printf("Please enter Y, y, N, or n.\n");
    else
      break;
  }
  (void) ichar_done();               /* assume worked */
  return answer == 'y';
}

int main()
{
  if (yesorno("What's your answer? "))
    printf("You answered YES!\n");
  else
    printf("You answered NO!\n");
  return 0;
}
```

screen.h

(Figure 20.26 – Page 528)

```
/*
 * Prototypes for direct screen access.
 */
#if defined(__MSDOS__)
#define display_init()    display_clear()
#else
void display_init(void);
#endif

void display_clear(void);
void display_goto(int, int);
```

dosscr.c

(Figure 20.27 – Page 529)

```c
/*
 * Screen output operations (for MS-DOS).
 *   display_clear - clear the screen.
 *   display_goto - move to a particular x,y location.
 */
#include <stdio.h>
#include "screen.h"

#if !defined(__MSDOS__)
#error This file should only be compiled under MS-DOS
#else
#include <dos.h>
void display_clear(void)
{
  union REGS regs;

  regs.h.ah = 0;                     /* OP: set text mode */
  regs.h.al = 2;                     /* clear as side effect */
  int86(0x10, &regs, &regs);
}

void display_goto(int row, int col)
{
  union REGS regs;

  regs.h.ah = 2;                     /* OP: set cursor position */
  regs.h.bh = 0;                     /* page to move to (always 0) */
  regs.h.dl = col;                   /* row to move to */
  regs.h.dh = row;                   /* column to move to */
  int86(0x10, &regs, &regs);
}
#endif
```

unixscr.c

(Figure 20.28 – Page 530)

```
/*
 * Screen output operations (for UNIX).
 *   display_init - initialize display.
 *   display_clear - clear display.
 *   display_goto - tmove to a particular x, y coordinate.
 */
#include <stdio.h>
#include "screen.h"

#if !defined(unix)
#error This file can only be compiled under UNIX
#else
#include <stdlib.h>                     /* for getenv */

extern int tgetent(const char *, const char *);
extern char *tgetstr(const char *, char **);
extern int tputs(const char *, int, int(*)(int));
extern char *tgoto(const char *, int, int);

static char *tn = NULL;                 /* terminal name */
static char tc[1024];                   /* terminal controls */
static char *ptr = tc;                  /* ptr to terminal controls */
static char *clearscreen;               /* clear screen chars */
static char *movecursor;                /* move cursor chars */
static char td[1024];                   /* termcap description */
```

unixscr.c

(Figure 20.28 – Page 530)

```
void display_clear(void)
{
  printf(clearscreen);
}

void display_init(void)
{
  if ((tn = getenv("TERM")) != NULL && tgetent(td, tn) == 1)
  {
    clearscreen = tgetstr("cl", &ptr);
    movecursor = tgetstr("cm", &ptr);
  }
  display_clear();
}

static int putcontrol(int c)
{
  return putchar(c);             /* need function, not macro */
}

void display_goto(int r, int c)
{
  (void) tputs(tgoto(movecursor, c, r), 1, putcontrol);
}
#endif
```

outworld.c

(Figure 20.29 – Page 531)

```
/*
 * Print generation using direct cursor addressing.
 */
#include <stdio.h>
#include "life.h"
#include "screen.h"

#define HEADER_ROW       0   /* row containing header */
#define BORDER_ROW       1   /* row containing top border */
#define BORDER_COL       0   /* col containing left border */
#define TOP_CELL_ROW     2   /* first row with cells */
#define LEFT_CELL_COL    1   /* first column with cells */

static void display_char(int row, int col, int n, char ch)
{
  for (display_goto(row,col); n-- > 0; putchar(ch))
    ;
}
```

outworld.c

(Figure 20.29 – Page 531)

```
void PutWorld(PTR_TO_CONST_WORLD world)
{
  int c, r;
  static int printed = FALSE;      /* first printing? */
  extern int Rows, Cols, Gen, EndGen;

  if (!printed)
  {
    display_init();
    display_char(BORDER_ROW, BORDER_COL, Cols + 2, BORDER);
    display_char(BORDER_ROW + Rows + 1, BORDER_COL,
                 Cols + 2, BORDER);
    for (r = TOP_CELL_ROW; r <=  TOP_CELL_ROW + Rows; r++)
    {                                   /* other borders */
      display_char(r, BORDER_COL, 1, BORDER);
      display_char(r, LEFT_CELL_COL + Cols, 1, BORDER);
    }
    printed = TRUE;
  }
  display_goto(HEADER_ROW, BORDER_COL);
  printf("Generation %i out of %i", Gen, EndGen);
  for (r = TOP_CELL_ROW; r < TOP_CELL_ROW + Rows; r++)
  {
    display_goto(r, LEFT_CELL_COL);
    for (c = 1; c <= Cols; c++)
      putchar(world[r][c] ? MARKER : ' ');
  }
  display_goto(BORDER_ROW + Rows + 3, BORDER_COL);
}
```

CHAPTER SUMMARY

- An important goal is to maximize the portability of the programs we write. This involves ensuring that our programs can be compiled under a variety of compilers and run on a variety of different machines, as well as under the control of different operating systems and with different I/O devices.

- When our programs must also compile on non-ANSI compilers, our biggest worries are function prototypes, generic pointers, and missing header files.

- The biggest difference between ANSI compilers is in the lengths of external names that are allowed. We use the preprocessor to define short names for any long, potentially conflicting function names.

- The biggest difference between operating systems is in the low-level system calls they provide. We need to define higher-level functions that internally use the appropriate low-level calls.

- There are a variety of different machine dependencies that can creep into C programs. But to write portable code we have to be careful to avoid using these troublesome constructs.