Dr. Estell's C Programs and Examples

Version 5.0 - 14 June 1996

```
/*
 *  helloworld.c  -  John K. Estell  -  6 April 1994
 *  This is the infamous "hello world" program traditionally shown as
 *  the first example of a C program.  It has been slightly modified
 *  to illustrate some other points about the language.
 *  By the way, this is an example of a header comment.
 *  Header comments should include: filename, author, date, and an
 *  overview of the program.  Version histories may also be included.
 *  Please note that this is written in the standard comment style.
 */

/*
 *  The following is a pre-processor instruction that is used to allow
 *  us to access the I/O functions in the stdio header file.
 */

#include <stdio.h>

/*
 *  C programs always start their execution with the main() function.
 *  This function must appear in all C programs.  The parentheses
 *  indicate that this is a function.  The "void" inside the parentheses
 *  indicate that no parameters are being passed to the function.  The "int" at
 *  the front indicates the returned data type, which is an integer.
 *
 *  Usually you'll see just "main()" - no int, no void.
 */

int main(void)
{               /*  Braces are used to enclose the body of a function.  */

   int num;     /* this declares a variable of type integer */

   num = 1;     /* assignment */

   /*
    *  The following statements are used to write to the standard output.
    *  The '\n' is used to represent the newline character.
    *  The '%d' is used to specify the printing out of an integer value.
    */

   printf("Hello world!!!\n");
   printf("I think my department is #%d.\n", num);

   /*
    *  The return statement ends the function and returns the specified
    *  value to the calling routine.  This statement can be left out of
    *  the main() function without any harm if so desired.  It must appear
    *  in all other functions that return a value.
    */

   return 0;
}
```

example of compilation errors

```
thor> cat errors.c
/*
 *  errors.c - name withheld by request - 7 April 1994
 *  This program has errors in it - will show results from
 *  gcc compilation using script...
 */

#include <stdio.h>

main()
(
  int n, int n2, int n3;

  n = 5;
  n2 = n * n;
  n3 = n2 * n2
  printf("n = %d, n squared = %d, n cubed = %d/n", n, n2, n3);
)

thor> gcc errors.c
errors.c:13: parse error before 'n'
errors.c:16: 'main' declared as function returning a function
errors.c:16: warning: data definition has no type or storage class

thor> cat -n errors.c
     1  /*
     2   *  errors.c - name withheld by request - 7 April 1994
     3   *  This program has errors in it - will show results from
     4   *  gcc compilation using script...
     5   */
     6
     7  #include <stdio.h>
     8
     9  main()
    10  (
    11    int n, int n2, int n3;
    12
    13    n = 5;
    14    n2 = n * n;
    15    n3 = n2 * n2
    16    printf("n = %d, n squared = %d, n cubed = %d/n", n, n2, n3);
    17  )
```

```
thor> cat errors1.c
/*
 *  errors1.c - name withheld by request - 7 April 1994
 *  This program still has errors in it - will show results from
 *  gcc compilation using script...
 */

#include <stdio.h>

main()
{
  int n, int n2, int n3;

  n = 5;
  n2 = n * n;
  n3 = n2 * n2
  printf("n = %d, n squared = %d, n cubed = %d/n", n, n2, n3);
}


thor> gcc errors1.c
errors1.c: In function 'main':
errors1.c:11: parse error before 'int'
errors1.c:14: 'n2' undeclared (first use this function)
errors1.c:14: (Each undeclared identifier is reported only once
errors1.c:14: for each function it appears in.)
errors1.c:15: 'n3' undeclared (first use this function)
errors1.c:16: parse error before 'printf'


thor> cat -n errors1.c
     1  /*
     2   *  errors.c - name withheld by request - 7 April 1994
     3   *  This program has errors in it - will show results from
     4   *  gcc compilation using script...
     5   */
     6
     7  #include <stdio.h>
     8
     9  main()
    10  {
    11    int n, int n2, int n3;
    12
    13    n = 5;
    14    n2 = n * n;
    15    n3 = n2 * n2
    16    printf("n = %d, n squared = %d, n cubed = %d/n", n, n2, n3);
    17  }
```

```
thor> cat errors2.c
/*
 *  errors2.c - name withheld by request - 7 April 1994
 *  This program still has errors in it - will show results from
 *  gcc compilation using script...
 */

#include <stdio.h>

main()
{
  int n, n2, n3;

  n = 5;
  n2 = n * n;
  n3 = n2 * n2;
  printf("n = %d, n squared = %d, n cubed = %d/n", n, n2, n3);
}


thor> gcc errors2.c
thor> a.out
n = 5, n squared = 25, n cubed = 625/nthor>

thor> cat -n errors2.c
     1  /*
     2   *  errors.c - name withheld by request - 7 April 1994
     3   *  This program has errors in it - will show results from
     4   *  gcc compilation using script...
     5   */
     6
     7  #include <stdio.h>
     8
     9  main()
    10  {
    11    int n, n2, n3;
    12
    13    n = 5;
    14    n2 = n * n;
    15    n3 = n2 * n2;
    16    printf("n = %d, n squared = %d, n cubed = %d/n", n, n2, n3);
    17  }
```

```
thor> cat errors3.c
/*
 *  errors3.c - name withheld by request - 7 April 1994
 *  This program no longer has errors in it - will show results from
 *  gcc compilation using script...
 */

#include <stdio.h>

main()
{
  int n, n2, n3;

  n = 5;
  n2 = n * n;
  n3 = n * n2;
  printf("n = %d, n squared = %d, n cubed = %d\n", n, n2, n3);
}


thor> gcc errors3.c
thor> a.out
n = 5, n squared = 25, n cubed = 125
thor>
thor> cat dbxdemo.c
/*
 *  dbxdemo.c  -  John K. Estell  -  15 June 1995
 *  This is a simple program constructed for showing off the
 *  use of the dbx debugging facility.  To use the debugger,
 *  you need to use the -g option when compiling:
 *
 *    {jupiter} gcc -g -o dbxdemo dbxdemo.c
 *
 *  Before using the debugger, print out a listing of your
 *  program using cat -n to get the line numbers:
 *
 *    {jupiter} cat -n dbxdemo.c | lpr
 *
 */

#include <stdio.h>

/* note that we can initialize variables when they are declared */

main()
{
   int count = 0, sum = 0, value;
   char ch;
   float average;
   /* read in values until a zero is entered, then print average */

   while (1)
   {
      printf("Enter integer: ");
      scanf("%d", &value);        /* input function */
      if (value == 0)
         break;       /* allows us to exit the loop */
      sum = sum + value;
      count++;
   }

   average = sum / count;  /* note that this gives a truncated answer */
   printf("The average of the values is %f\n", average);

   /* get a character and print out its ASCII value */

   printf("Enter an integer from 32 to 127: ");
   scanf("%d", &value);
   ch = (char) value;
   printf("The ASCII value for '%c' is %d.\n", ch, ch);
}
```

```
thor> cat -n dbxdemo.c
     1  /*
     2   *  dbxdemo.c  -  John K. Estell  -  15 June 1995
     3   *  This is a simple program constructed for showing off the
     4   *  use of the dbx debugging facility.  To use the debugger,
     5   *  you need to use the -g option when compiling:
     6   *
     7   *    {jupiter} gcc -g -o dbxdemo dbxdemo.c
     8   *
     9   *  Before using the debugger, print out a listing of your
    10   *  program using cat -n to get the line numbers:
    11   *
    12   *    {jupiter} cat -n dbxdemo.c | lpr
    13   *
    14   */
    15
    16  #include <stdio.h>
    17
    18  /* note that we can initialize variables when they are declared */
    19
    20  main()
    21  {
    22     int count = 0, sum = 0, value;
    23     char ch;
    24     float average;
    25
    26     /* read in values until a zero is entered, then print average */
    27
    28     while (1)
    29     {
    30        printf("Enter integer: ");
    31        scanf("%d", &value);        /* input function */
    32        if (value == 0)
    33           break;       /* allows us to exit the loop */
    34        sum = sum + value;
    35        count++;
    36     }
    37
    38     average = sum / count;   /* note that this gives a truncated answer */
    39     printf("The average of the values is %f\n", average);
    40
    41     /* get a character and print out its ASCII value */
    42
    43     printf("Enter an integer from 32 to 127: ");
    44     scanf("%d", &value);
    45     ch = (char) value;
    46     printf("The ASCII value for '%c' is %d.\n", ch, ch);
    47  }
```

```
thor> gcc -g -o dbxdemo dbxdemo.c
thor> dbx dbxdemo
Reading symbolic information...
Read 162 symbols
(dbx) run
Running: dbxdemo
Enter integer: 4
Enter integer: 5
Enter integer: 0
The average of the values is 4.000000
Enter an integer from 32 to 127: 65
The ASCII value for 'A' is 65.

execution completed, exit code is 1
program exited with 1
(dbx) trace
(3) trace
(dbx) run
Running: dbxdemo
trace:     21   {
trace:     28      while (1)
trace:     30         printf("Enter integer: ");
trace:     31         scanf("%d", &value);        /* input function */
Enter integer: 4
trace:     32         if (value == 0)
trace:     34         sum = sum + value;
trace:     35         count++;
trace:     36      }
trace:     28      while (1)
trace:     30         printf("Enter integer: ");
trace:     31         scanf("%d", &value);        /* input function */
Enter integer: 5
trace:     32         if (value == 0)
trace:     34         sum = sum + value;
trace:     35         count++;
trace:     36      }
trace:     28      while (1)
trace:     30         printf("Enter integer: ");
trace:     31         scanf("%d", &value);        /* input function */
Enter integer: 0
trace:     32         if (value == 0)
trace:     33            break;       /* allows us to exit the loop */
trace:     38      average = sum / count;   /* note that this gives a truncated answer */
trace:     39      printf("The average of the values is %f\n", average);
The average of the values is 4.000000
trace:     43      printf("Enter an integer from 32 to 127: ");
trace:     44      scanf("%d", &value);
Enter an integer from 32 to 127: 65
trace:     45      ch = (char) value;
trace:     46      printf("The ASCII value for '%c' is %d.\n", ch, ch);
The ASCII value for 'A' is 65.
trace:     47   }
program exited with 1
```

```
(dbx) status
(3) trace
(dbx) delete 3
(dbx) status
(dbx) stop at 32
(11) stop at "/home/jupiter/estell/dbxdemo.c":32
(dbx) stop at 45
(12) stop at "/home/jupiter/estell/dbxdemo.c":45
(dbx) run
Running: dbxdemo
Enter integer: 4
stopped in main at line 32 in file "/home/jupiter/estell/dbxdemo.c"
   32         if (value == 0)
(dbx) step
stopped in main at line 34 in file "/home/jupiter/estell/dbxdemo.c"
   34         sum = sum + value;
(dbx) cont
Enter integer: 5
stopped in main at line 32 in file "/home/jupiter/estell/dbxdemo.c"
   32         if (value == 0)
(dbx) step
stopped in main at line 34 in file "/home/jupiter/estell/dbxdemo.c"
   34         sum = sum + value;
(dbx) cont
Enter integer: 0
stopped in main at line 32 in file "/home/jupiter/estell/dbxdemo.c"
   32         if (value == 0)
(dbx) step
stopped in main at line 33 in file "/home/jupiter/estell/dbxdemo.c"
   33            break;       /* allows us to exit the loop */
(dbx) step
stopped in main at line 38 in file "/home/jupiter/estell/dbxdemo.c"
   38      average = sum / count;    /* note that this gives a truncated answer */
(dbx) cont
The average of the values is 4.000000
Enter an integer from 32 to 127: 65
stopped in main at line 45 in file "/home/jupiter/estell/dbxdemo.c"
   45      ch = (char) value;
(dbx) print ch
ch = '\0'
(dbx) step
stopped in main at line 46 in file "/home/jupiter/estell/dbxdemo.c"
   46      printf("The ASCII value for '%c' is %d.\n", ch, ch);
(dbx) print ch
ch = 'A'
(dbx) cont
The ASCII value for 'A' is 65.

execution completed, exit code is 1
program exited with 1
```

```
(dbx) status
(11) stop at "/home/jupiter/estell/dbxdemo.c":32
(12) stop at "/home/jupiter/estell/dbxdemo.c":45
(dbx) delete 11
(dbx) delete 12
(dbx) stop at 32 if value == 0
(14) stop  at "/home/jupiter/estell/dbxdemo.c":32 if value == 0
(dbx) run
Running: dbxdemo
Enter integer: 4
Enter integer: 5
Enter integer: 0
stopped in main at line 32 in file "/home/jupiter/estell/dbxdemo.c"
   32         if (value == 0)
(dbx) step
stopped in main at line 33 in file "/home/jupiter/estell/dbxdemo.c"
   33            break;       /* allows us to exit the loop */
(dbx) step
stopped in main at line 38 in file "/home/jupiter/estell/dbxdemo.c"
   38      average = sum / count;   /* note that this gives a truncated answer */
(dbx) step
stopped in main at line 39 in file "/home/jupiter/estell/dbxdemo.c"
   39      printf("The average of the values is %f\n", average);
(dbx) print sum
sum = 9
(dbx) print count
count = 2
(dbx) print sum / count
sum/count = 4.5
(dbx) print average
average = 4.0
(dbx) cont
The average of the values is 4.000000
Enter an integer from 32 to 127: 65
The ASCII value for 'A' is 65.

execution completed, exit code is 1
program exited with 1
(dbx) quit
thor>
```

Brief dbx command summary:

`run`	Starts execution from beginning of program
`step`	Executes the next statement
`cont`	Resumes execution
`status`	Prints out all pending dbx instructions (trace, stop, etc.)
`trace`	Traces line by line the execution of the program
`where`	Displays current location
`stop at #`	Insert breakpoint - execution will stop before executing statement at specified line number
`print expr`	Evaluates expression, then displays the result
`delete #`	Removes specified dbx instruction - number is obtained from first value of status output

Notes:

To effectively use `dbx`, make sure that you have a line numbered listing of your source file:

```
1. cat -n sourcefile.c | lpr
2. gcc -g -o sourcefile sourcefile.c
3. dbx sourcefile
```

Be aware that when a program is compiled with the `-g` option, it will behave differently than under normal compilation. For example, all integers, unless otherwise specified, are automatically initialized to zero. Under normal situations, no initialization is performed. This can cause a program that is buggy to work correctly when compiled for the debugger!

```
/*
 *  int.c  -  John K. Estell  -  9 April 1994
 *  Integer examples
 */

#include <stdio.h>

main()
{
   int num1;
   int num2 = 2;

   printf("num1 = %d  num2 = %d\n\n", num1, num2);

   printf("Enter num1: ");
   scanf("%d", &num1);
   printf("num1 = %d\n\n", num1);

   printf("2^%2d = %5d\n", 3, 8);
   printf("2^%2d = %5d\n", 4, 16);
   printf("2^%2d = %5d\n", 6, 64);
   printf("2^%2d = %5d\n", 8, 256);
   printf("2^%2d = %5d\n", 10, 1024);
   printf("2^%2d = %5d\n", 16, 65536);
}

thor> gcc int.c
thor> a.out
num1 = 8192  num2 = 2

Enter num1: 86
num1 = 86

2^ 3 =     8
2^ 4 =    16
2^ 6 =    64
2^ 8 =   256
2^10 =  1024
2^16 = 65536
thor>
```

```
/*
 *  beeper.c  -  John K. Estell  -  9 April 1994
 *  Demonstration of data types
 */

#include <stdio.h>

main()
{
   float speed, distance, time;    /* floating-point variable declaration */
   char  beep;                     /* character variable declaration */

   /* note that a character is enclosed within single quotes */

   beep = '\a';          /* define beep to be the "audible alert" character */

   /*
    *  tell the user about the program.
    *  note that strings are enclosed within double quotes.
    */

   printf("       ACME Software Development Company\n");
   printf("Road Runner Apprehension Probability Calculator\n");

   /* get information from the user */

   printf("\n");
   printf("Distance in meters to the Road Runner: ");
   scanf("%f", &distance);

   printf("Maximum speed in km/h of your Foyt-Coyote engine: ");
   scanf("%f", &speed);

   /* calculate the best-case time for reaching the coyote chow */

   speed = 1000.0 * speed / 3600.0;   /* convert from km/h to m/s */
   time = distance / speed;

   /* print out the results */

   printf("Time to reach Road Runner's position is %.2f seconds.\n", time);
   printf("... but he'll be gone!!! %c%c\n", beep, beep);
   printf("Probability of having Road Runner Stew is %d%%.\n", 0);
}
```

```
thor> gcc -o beeper beeper.c
thor> beeper
       ACME Software Development Company
Road Runner Apprehension Probability Calculator

Distance in meters to the Road Runner: 1000
Maximum speed in km/h of your Foyt-Coyote engine: 200
Time to reach Road Runner’s position is 18.00 seconds.
... but he’ll be gone!!!
Probability of having Road Runner Stew is 0%.
thor> beeper
       ACME Software Development Company
Road Runner Apprehension Probability Calculator

Distance in meters to the Road Runner: 20
Maximum speed in km/h of your Foyt-Coyote engine: 400
Time to reach Road Runner’s position is 0.18 seconds.
... but he’ll be gone!!!
Probability of having Road Runner Stew is 0%.
thor>
```

size of data types

```
/*
 *  typesize.c - prints out type sizes
 *  From C Primer Plus book
 *  Please note that there are many uses for the sizeof() function as, since data sizes are
 *  machine specific, the ability to obtain the actual size allows programs to be portable.
 *  This program will report different results on different systems!
 */

#include <stdio.h>

main()
{
   printf("Type short int has a size of %d bytes.\n", sizeof(short int));
   printf("Type int has a size of %d bytes.\n", sizeof(int));
   printf("Type long int has a size of %d bytes.\n", sizeof(long int));
   printf("Type char has a size of %d bytes.\n", sizeof(char));
   printf("Type float has a size of %d bytes.\n", sizeof(float));
   printf("Type double has a size of %d bytes.\n", sizeof(double));
}

thor> gcc typesize.c
thor> a.out
Type short int has a size of 2 bytes.
Type int has a size of 4 bytes.
Type long int has a size of 4 bytes.
Type char has a size of 1 bytes.
Type float has a size of 4 bytes.
Type double has a size of 8 bytes.
thor>
```

```
/*
 *  define.c  -  John K. Estell  -  14 April 1994
 *  Simple example of the use (and abuse?) of the define pre-processor
 *  statement.  All define statements are in the define.h header file,
 *  to demonstrate how to include your own files.  Both define.c and
 *  define.h must be in the same working directory when compiled.
 */

#include <stdio.h>        /* include system header file "/usr/include/stdio.h" */
#include "define.h"       /* include user's header file "./define.h"           */

main()
{
   float radius, area, volume, cost;

   /* Tell the user who and what we are, then get the appropriate info. */

   output("Pie in the Face Corporation");
   output("Let's us calculate what size pie you need!");
   newline;
   inquiry("Enter radius (in inches) of the victim's face: ", radius);

   /* Perform the calculations. */

   area = PI * radius * radius;
   volume = PIE_TIN_DEPTH * area;
   cost = UNIT_VOLUME_COST * volume;

   /* Display the results. */

   newline;
   output("*** RESULTS ***");
   newline;
   output("YES!  We can make a pie for YOU!");
   newline;
   printf("The area of the pie is %.2f square inches\n", area);
   printf("The volume of the pie is %.2f cubic inches.\n", volume);
   printf("The cream filling costs $%.2f/in^3.\n", UNIT_VOLUME_COST);
   printf("Your cost is $%.2f.\n", cost);
   newline;
}
```

```
/*
 *  define.h  -  John K. Estell  -  14 April 1994
 *  This file contains define statements for use with the demonstration
 *  program define.c
 */

/* simple constants */

#define PI                 3.14159
#define PIE_TIN_DEPTH      2.5
#define UNIT_VOLUME_COST   0.86

/* fancier defines - not recommended for casual use as it can promote poor style. */

#define newline                    printf("\n")
#define output( text )             printf( text ), newline
#define inquiry( prompt, answer )  printf( prompt ), scanf("%f", &answer)
```

```
thor> gcc -o define define.c
thor> define
Pie in the Face Corporation
Let’s us calculate what size pie you need!

Enter radius (in inches) of the victim’s face: 4.2

*** RESULTS ***

YES!  We can make a pie for YOU!

The area of the pie is 55.59 square inches
The volume of the pie is 138.99 cubic inches.
The cream filling costs $0.86/in^3.
Your cost is $119.53.
```

```
thor> vi define.h

    ... edited the file to reflect correct cost of pie filling ...

thor> cat define.h

/*
 *  define.h  -  John K. Estell  -  14 April 1994
 *  This file contains define statements for use with the demonstration
 *  program define.c
 */

/* simple constants */

#define PI                 3.14159
#define PIE_TIN_DEPTH      2.5
#define UNIT_VOLUME_COST   0.086

/* fancier defines */

#define newline                    printf("\n")
#define output( text )             printf( text ), newline
#define inquiry( prompt, answer )  printf( prompt ), scanf("%f", &answer)

thor> gcc -o define define.c
thor> define
Pie in the Face Corporation
Let's us calculate what size pie you need!

Enter radius (in inches) of the victim's face: 4.2

*** RESULTS ***

YES!  We can make a pie for YOU!

The area of the pie is 55.59 square inches
The volume of the pie is 138.99 cubic inches.
The cream filling costs $0.09/in^3.
Your cost is $11.95.
```

problems with the scanf input function

```
/*
 *  input1.c  -  John K. Estell  -  17 April 1994
 *  Demo for what can go wrong with scanf
 */

#include <stdio.h>

main()
{
   int val;
   int status = 0;

   /*
    *  the scanf below returns one of the following:
    *  -1 if nothing is entered, 0 if entry is not "correctly spelled", 1 if an integer.
    */

   while (status < 1)
   {
      printf("Enter an integer: ");
      status = scanf("%d", &val);

      if (status == 0)
         printf("That was not an integer!\n");
   }

   printf("The integer entered was %d.\n", val);
}
thor> gcc -o input1 input1.c
thor> input1
Enter an integer: 24
The integer entered was 24.
thor> input1
Enter an integer:
Enter an integer:      45      (scanf skips over leading whitespace)
The integer entered was 45.
thor> input1
Enter an integer:  34  wd      (scanf uses whitespace as a delimiter)
The integer entered was 34.
thor> input1
Enter an integer:  56moo       (scanf will stop once a non-integer character is
encountered)
The integer entered was 56.
thor> input1
Enter an integer: hi           (scanf fails - input is left in the system input buffer...)
That was not an integer!
Enter an integer: That was not an integer!
Enter an integer: That was not an integer!
Enter an integer: That was not an integer!
Enter an integer^C
thor>
```

{Note: when using `script` ***this actually resulted in an infinite loop! When not in*** `script` ***the control-C did stop the runaway output.}***

```
/*
 *  input2.c  -  John K. Estell  -  17 April 1994
 *  Demonstration of using gets() and sscanf() to
 *  avoid scanf() problems.
 */

#include <stdio.h>

main()
{
   int val;
   int status = 0;
   char inbuf[133];

   while (status < 1)
   {
      printf("Enter an integer: ");
      gets(inbuf);            /* reads in one line of characters from the standard input */
      status = sscanf(inbuf, "%d", &val);  /* uses string as input to scanf-type function */

      if (status == 0)
         printf("That was not an integer!\n");
   }

   printf("The integer entered was %d.\n", val);
}
thor> gcc -o input2 input2.c
thor> input2
Enter an integer: 23
The integer entered was 23.
thor> input2
Enter an integer:      86
The integer entered was 86.
thor> input2
Enter an integer: 99 lalala    (whitespace still acts as a delimiter)
The integer entered was 99.
thor> input2
Enter an integer: an integer   (no problem when sscanf fails - we just discard the line)
That was not an integer!
Enter an integer: hello        (overwriting of the character buffer works fine for now...)
That was not an integer!
Enter an integer: 23e4         (floating point notation doesn’t work with integer input)
The integer entered was 23.
thor>
```

integer division and modulo arithmetic

```
/*
 *  fractions.c  -  John K. Estell  -  17 April 1994
 *  Perform division, reporting the result as a mixed fraction.
 */

#include <stdio.h>

main()
{
   int divisor, dividend, quotient, remainder;
   int valid_input = 0;
   char inbuf[133];

   /* get the divisor and the dividend */

   while (valid_input < 1)
   {
      printf("Enter the dividend (the number to be divided): ");
      gets(inbuf);
      valid_input = sscanf(inbuf, "%d", &dividend);
   }

   valid_input = 0;  /* reset the flag */

   while (valid_input < 1)
   {
      printf("Enter the divisor (the number to divide by): ");
      gets(inbuf);
      valid_input = sscanf(inbuf, "%d", &divisor);

      /* check for an attempt to divide by zero */

      if (divisor == 0)
      {
         printf("Division by zero is not allowed.\n");
         valid_input = 0;
      }
   }

   /* perform the division */

   quotient = dividend / divisor;     /* integer division yields only the quotient */
   remainder = dividend % divisor;    /* % is the modulo operator - yields the remainder */

   /* print the results */

   printf("%d / %d = %d and %d/%d\n",
          dividend, divisor, quotient, remainder, divisor);
}
```

```
thor> gcc -o fractions fractions.c
thor> fractions
Enter the dividend (the number to be divided): 17
Enter the divisor (the number to divide by): 5
17 / 5 = 3 and 2/5
thor> fractions
Enter the dividend (the number to be divided): sfisdfhdshf
Enter the dividend (the number to be divided): 6
Enter the divisor (the number to divide by): 3
6 / 3 = 2 and 0/3
thor> fractions
Enter the dividend (the number to be divided): 200
Enter the divisor (the number to divide by): 0
Division by zero is not allowed.
Enter the divisor (the number to divide by): 17
200 / 17 = 11 and 13/17
thor>
```

reverse.c - another integer arithmetic example

```c
/*
 *  reverse.c  -  John K. Estell  -  17 April 1994
 *  Given an integer input, this will print out the reverse of
 *  the number (e.g. 321 becomes 123)
 */

#include <stdio.h>

main()
{
   int num, rev = 0;
   int valid_input = 0;
   char inbuf[133];

   /* Get an integer from the user - use gets/sscanf for idiot-proofing */

   while (valid_input < 1)
   {
      printf("Enter an integer: ");
      gets(inbuf);
      valid_input = sscanf(inbuf, "%d", &num);
   }

   /*
    *  Reverse the number: multiply rev by ten to shift digits to the left,
    *  add the units digit from num, then integer divide num by ten to
    *  shift digits to the right.
    */

   while (num != 0)
   {
      rev *= 10;         /* same as writing  rev = rev * 10; */
      rev += num % 10;   /* same as writing  rev = rev + (num % 10); */
      num /= 10;         /* same as writing  num = num / 10; */
   }

   /* Print out the result */

   printf("Reverse number is %d.\n", rev);
}
```

```
thor> gcc -o reverse reverse.c
thor> reverse
Enter an integer: 12340
Reverse number is 4321.
thor> reverse
Enter an integer: R
Enter an integer: 998642
Reverse number is 246899.
thor> reverse
Enter an integer: -9753
Reverse number is -3579.
thor>
```

```
/*
 *  math.c  -  John K. Estell  -  19 April 1994
 *  Show off different methods of using math functions.
 *  Please note that all floating-point math functions use
 *  the "double" floating-point type.
 *  Compile with: gcc -o math math.c -lm
 *  The -lm will tell gcc to link in the math library when forming the executable.
 */

#include <stdio.h>
#include <math.h>

main()
{
   int    int_value;
   float  float_value;
   double dbl_value;
   char   inbuf[133];

   /*
    *  make up some excuse to obtain input from the user
    *  assuming correct input for purposes of this example; obviously not
    *  something you would want to do in a real program!
    */

   printf("\nPlease enter a positive integer: ");
   gets(inbuf);
   sscanf(inbuf, "%d", &int_value);

   printf("Please enter a value between 0 and 1: ");
   gets(inbuf);
   sscanf(inbuf, "%f", &float_value);

   /*
    *  now do some number crunching - M_PI is a constant defined in
    *  math.h.  To see what constants are available please look at
    *  the file  /usr/include/math.h
    */

   dbl_value = asin( (double) float_value );   /* arguments are usually of type double */
   dbl_value *= 180.0 / M_PI;                  /* converting from radians to degrees */

   /* print out stuff... */

   printf("\n");
   printf("The square root of %d is %f.\n", int_value, sqrt( (double) int_value ));
   printf("The arcsine of %.3f is %.1f degrees.\n", float_value, dbl_value);
   printf("1.0 divided by 0.0 is %f.\n", (1.0 / 0.0));
   printf("-1.0 divided by 0.0 is %f.\n", (-1.0 / 0.0));
   printf("0.0 divided by 0.0 is %f.\n", (0.0 / 0.0));
}
```

```
thor> gcc -o math math.c -lm
thor> math

Please enter a positive integer: 2
Please enter a value between 0 and 1: .707

The square root of 2 is 1.414214.
The arcsine of 0.707 is 45.0 degrees.
1.0 divided by 0.0 is Inf.      (printf displays "Inf" for the value infinity)
-1.0 divided by 0.0 is -Inf.
0.0 divided by 0.0 is NaN.      (printf displays "NaN" for mathematically undefined values)
thor>
```

random number generation

```
/*
 *  random.c  -  John K. Estell  -  22 April 1994
 *  Illustrates use of the random number generator
 *  by simulating a dice toss.
 */

#include <stdio.h>
#include <stdlib.h>   /* for the rand() function */
#include <limits.h>   /* for the INT_MAX value   */

/*
 *  define RAND_MAX constant as largest positive value of type int
 *  note that this is not defined on some systems, so check to see if it’s there...
 */

#ifndef RAND_MAX
#define RAND_MAX  INT_MAX
#endif

main()
{
   int ran1, ran2;  /* variables used for the random number   */
   int die1, die2;  /* variables that will hold final results */
   int seed;        /* seed value for random number */

   /*
    *  get random number seed from user - later we’ll learn better methods
    *  for getting seed values automatically.
    */

   printf("Enter an integer: ");
   scanf("%d", &seed);
   srand(seed);

   /* get the random numbers: range of returned integer is from 0 to RAND_MAX */

   ran1 = rand();
   ran2 = rand();
   if (ran1 == RAND_MAX)   /* done to insure that ran1 / RAND_MAX is < 1 */
      ran1--;
   if (ran2 == RAND_MAX)
      ran2--;

   /* convert the integer range [0,RAND_MAX) to [1,6] - use int cast to truncate result */

   die1 = (int) ( 6.0 * ((float) ran1 / (float) RAND_MAX) + 1 );
   die2 = (int) ( 6.0 * ((float) ran2 / (float) RAND_MAX) + 1 );

   /* print the results of the throw */

   printf("You have rolled a %d and a %d.\n", die1, die2);
}
```

```
thor> gcc -o random random.c
thor> random
Enter an integer: 1
You have rolled a 4 and a 2.
thor> random
Enter an integer: 1          (use of the same seed value results in the same outcome)
You have rolled a 4 and a 2.
thor> random
Enter an integer: 2
You have rolled a 1 and a 5.
thor> random
Enter an integer: 427
You have rolled a 3 and a 6.
thor> random
Enter an integer: 86
You have rolled a 2 and a 3.
thor> random
Enter an integer: 9876
You have rolled a 6 and a 6.
thor> random
Enter an integer: 4567
You have rolled a 5 and a 2.
thor>
```

The use of a pseudo-random number generator allows for duplication of results requiring "random" inputs by using the same seed value. If true randomness is required then one seeds the generator using something that varies, such as time. Using continued seeding, something close to true randomness can be approached.

bounded integer input routine

```c
/*
 *  iolib.h  -  John K. Estell  -  7 September 1995
 *  library of personal I/O routines
 */

#include <limits.h>

#define BUFFER_SIZE 133

int get_bounded_integer(int lower_bound, int upper_bound);

int get_bounded_integer(int lower_bound, int upper_bound)
{
   int user_input;
   int flag = 0;
   char buffer[BUFFER_SIZE];

   while (flag < 1)
   {
      printf("Enter an integer: ");
      fgets(buffer, BUFFER_SIZE, stdin);
      flag = sscanf(buffer, "%d", &user_input);

      if (flag == - 1)
         printf("Please enter an integer.\n");
      else if (flag == 0)
         printf("That's not an integer - please enter an integer.\n");
      else
      {
         if (user_input < lower_bound)
         {
            printf("Please enter an integer >= %d.\n", lower_bound);
            flag = 0;
         }
         else if (user_input > upper_bound)
         {
            printf("Please enter an integer <= %d.\n", upper_bound);
            flag = 0;
         }
      }
   }

   return user_input;
}
```

```
/*
 *  recurrence.c  -  John K. Estell  -  7 September 1995
 *  demonstration program for a solved recurrence relation.
 *
 *  compile: gcc -o recurrence recurrence.c -lm
 *
 *  Find recurrence relation for the number
 *  of ternary strings that do not contain two consecutive zeros.
 *
 *  Recurrence relation: a[n] = 2a[n-1] + 2a[n-2], n >= 2
 *  Initial conditions : a[0] = 1, a[1] = 3
 */

#include <stdio.h>
#include <math.h>
#include "iolib.h"

main()
{
   double root1, root2, alpha1, alpha2, result;
   int n;

   /* set up the constants */

   root1 = 1.0 + sqrt(3.0);
   root2 = 1.0 - sqrt(3.0);

   alpha1 = ( 2.0 + sqrt(3.0) ) / ( 2 * sqrt(3.0) );
   alpha2 = ( sqrt(3.0) - 2.0 ) / ( 2 * sqrt(3.0) );

   /* get the value of n */

   n = get_bounded_integer(0, INT_MAX);

   /* calculate the result and display answer */

   result = alpha1 * pow(root1, (double) n) + alpha2 * pow(root2, (double) n);

   printf("f(%d) = %f\n", n, result);
}
```

```
thor> gcc -o recurrence recurrence.c -lm
thor> recurrence
Enter an integer: 0
f(0) = 1.000000
thor> recurrence
Enter an integer: 2
f(2) = 8.000000
thor> recurrence
Enter an integer: 6
f(6) = 448.000000
thor> recurrence
Enter an integer: 10
f(10) = 24960.000000
```

function library for complex numbers

```
/*
 *  complexlib.h  -  John K. Estell  -  8 September 1995
 *  library for working with complex numbers
 */

typedef struct
{
  double real;
  double imag;
} complex;

complex add(complex addend, complex augend);
complex subtract(complex minuend, complex subtrahend);
complex multiply(complex multiplier, complex multiplicand);
complex power(complex base, int degree);
char *print_complex(complex value);

complex add(complex addend, complex augend)
{
   complex result;

   result.real = addend.real + augend.real;
   result.imag = addend.imag + augend.imag;

   return result;
}

complex subtract(complex minuend, complex subtrahend)
{
   complex result;

   result.real = minuend.real - subtrahend.real;
   result.imag = minuend.imag - subtrahend.real;

   return result;
}

complex multiply(complex multiplier, complex multiplicand)
{
   complex result;

   result.real = multiplier.real * multiplicand.real
                 - multiplier.imag * multiplicand.imag;
   result.imag = multiplier.real * multiplicand.imag
                 + multiplier.imag * multiplicand.real;

   return result;
}
```

```
complex power(complex base, int degree)
{
   complex result;
   int i;

   result.real = 1.0;
   result.imag = 0.0;

   for (i = 1; i <= degree; i++)
      result = multiply(result, base);

   return result;
}

char *print_complex(complex value)
{
   static char buffer[133];

   if (value.imag < 0.0)
      sprintf(buffer, "%f-%fi\0", value.real, -value.imag);
   else
      sprintf(buffer, "%f+%fi\0", value.real, value.imag);

   return buffer;
}
```

```
/*
 *  rec2.c  -  John K. Estell  -  8 September 1995
 *  demonstration program for a solved recurrence relation.
 *
 *  Section 5.2, Exercise 22:
 *
 *  Recurrence relation: a[n] = 2a[n-1] - 2a[n-2], n >= 2
 *  Initial conditions :  a[0] = 1, a[1] = 2
 */

#include <stdio.h>
#include "iolib.h"
#include "complexlib.h"

main()
{
   complex root1, root2, alpha1, alpha2, result;
   complex partial1, partial2;
   int n;

   /* set up the constants */

   root1.real = 1.0;
   root1.imag = 1.0;
   root2.real = 1.0;
   root2.imag = -1.0;

   alpha1.real = 0.5;
   alpha1.imag = -0.5;
   alpha2.real = 0.5;
   alpha2.imag = 0.5;

   /* get the value of n */

   n = get_bounded_integer(0, INT_MAX);

   /* calculate the result */

   partial1 = multiply(alpha1, power(root1, n));
   partial2 = multiply(alpha2, power(root2, n));
   result = add(partial1, partial2);

   /* display answer */

   printf("f(%d) = %s\n", n, print_complex(result));
}
```

```
thor> gcc -o rec2 rec2.c
thor> rec2
Enter an integer: -1
Please enter an integer >= 0.
Enter an integer:
Please enter an integer.
Enter an integer: an integer
That's not an integer - please enter an integer.
Enter an integer: 0
f(0) = 1.000000+0.000000i
thor> rec2
Enter an integer: 1
f(1) = 2.000000+0.000000i
thor> rec2
Enter an integer: 2
f(2) = 2.000000+0.000000i
thor> rec2
Enter an integer: 3
f(3) = 0.000000+0.000000i
thor> rec2
Enter an integer: 4
f(4) = -4.000000+0.000000i
thor> rec2
Enter an integer: 5
f(5) = -8.000000+0.000000i
thor> rec2
Enter an integer: 5
f(5) = -8.000000+0.000000i
thor> rec2
Enter an integer: 6
f(6) = -8.000000+0.000000i
thor> rec2
Enter an integer: 7
f(7) = 0.000000+0.000000i
thor> rec2
Enter an integer: 8
f(8) = 16.000000+0.000000i
thor> rec2
Enter an integer: 9
f(9) = 32.000000+0.000000i
thor> rec2
Enter an integer: 10
f(10) = 32.000000+0.000000i
thor>
```

```
/*
 *  rubberband.c - Jennyfur Xenon - 1 April 1994
 *  calculation program to determine and verify target parameters....
 *  Compile with: gcc -o rubberband rubberband.c -lm
 */

#include <stdio.h>
#include <math.h>

#define target_distance 30.0   /* professor is 30 feet away */
#define target_height   5.75   /* professor is 5'9" tall.... */

main()
{
  double velocity, x_vel, y_vel, theta, time, pi, height;

  /* Give target information and obtain attack parameters */

  printf("Target distance is %.1f feet away.\n", target_distance);
  printf("Target is %.2f feet tall.\n\n", target_height);

  printf("Enter angle of elevation in degrees: ");
  scanf("%lf", &theta);  /* "&lf" is used to read in values of type double */
  printf("Enter initial velocity in feet per second: ");
  scanf("%lf", &velocity);
  printf("Enter initial height in feet: ");
  scanf("%lf", &height);

  /* Calculate x and y velocity components */

  pi = 2.0 * asin(1.0);
  x_vel = velocity * cos( theta * pi / 180 );
  y_vel = velocity * sin( theta * pi / 180 );

  /* Calculate time to target and height of rubber band at that time */

  time = target_distance / x_vel;
  height = height + y_vel * time - 16.0 * time * time;

  /*
   *  Determine the results - professor has been hit if the height of the rubber band is
   *  between the professor's height and ground, assuming Jennyfur can shoot straight!
   */

  if ((height > 0.0) && (height < target_height))  /* "&&" is the logical AND operator */
  {
    printf("You hit the professor with the rubber band!!!\n");
    printf("YOU ARE THE REIGNING RUBBER BAND CHAMPION!!!\n");
  }
  else
    printf("You missed, furball!!!\n");

  printf("Height of rubber band at target was %.2f feet.\n", height);
}
```

```
thor> gcc -o rubberband rubberband.c  (when the -lm library is not specified....)
ld: Undefined symbol
   _cos
   _asin
   _sin
collect: /usr/bin/ld returned 2 exit status   (... you get the following error messages)
thor> gcc -o rubberband rubberband.c -lm
thor> rubberband
Target distance is 30.0 feet away.
Target is 5.75 feet tall.

Enter angle of elevation in degrees: 45
Enter initial velocity in feet per second: 10
Enter initial height in feet: 3
You missed, furball!!!
Height of rubber band at target was -255.00 feet.
thor> rubberband
Target distance is 30.0 feet away.
Target is 5.75 feet tall.

Enter angle of elevation in degrees: 30
Enter initial velocity in feet per second: 300
Enter initial height in feet: 3
You missed, furball!!!
Height of rubber band at target was 20.11 feet.
thor>

... and on the eighty-sixth try:

thor> rubberband
Target distance is 30.0 feet away.
Target is 5.75 feet tall.

Enter angle of elevation in degrees: 10
Enter initial velocity in feet per second: 53
Enter initial height in feet: 3
You hit the professor with the rubber band!!!
YOU ARE THE REIGNING RUBBER BAND CHAMPION!!!
Height of rubber band at target was 3.00 feet.
thor>
```

```
/*
 *  relational.c  -  John K. Estell  -  22 April 1994
 *  Perform relation tests for two values.
 */

#include <stdio.h>

main()
{
   int a, b;

   /* get the inputs */

   printf("Enter the value for the integer a: ");
   scanf("%d", &a);
   printf("Enter the value for the integer b: ");
   scanf("%d", &b);

   /* display the comparative results */

   printf("a <  b = %d\n", a < b );    /* less than */
   printf("a <= b = %d\n", a <= b );   /* less than or equal to */
   printf("a >  b = %d\n", a > b );    /* greater than */
   printf("a >= b = %d\n", a >= b );   /* greater than or equal to */
   printf("a == b = %d\n", a == b );   /* is equal to */
   printf("a != b = %d\n", a != b );   /* is not equal to */
   printf("    !a = %d\n", !a );       /* logical NOT */
   printf("    !b = %d\n", !b );
}

thor> gcc -o relational relational.c
thor> relational
Enter the value for the integer a: 1
Enter the value for the integer b: 22
a <  b = 1   (true: assigned value is 1; anything that’s non-zero is evaluated as true)
a <= b = 1
a >  b = 0   (false: assigned value is 0; 0 is evaluated as false)
a >= b = 0
a == b = 0
a != b = 1
    !a = 0
    !b = 0
thor>
```

```
/*
 *  arrayloop.c  -  John K. Estell  -  26 April 1994
 *  Demonstration of using loops to process an array
 */

#include <stdio.h>

/*
 *  The following constant is used to declare the size of the integer array.
 *  If the size later changes, then only the define statement needs to be edited
 *  instead of going all throughout the source code.
 */

#define SIZE 10

main()
{
   int   i,
         sum = 0,
         numbers[SIZE];  /* note that the array index range is from 0 to SIZE-1 */
   char  letters[99] = "The average of the integers entered is "; /* legal only here... */
   float result;

   /* get some numbers for the array */

   printf("Let's average %d integers, shall we?\n\n", SIZE);

   for (i = 0; i < SIZE; i++)             /* initialize i to 0 */
   {                                   /* LOOP: check here to see if i is less than SIZE */
      printf("Enter integer #%d: ", i+1); /* execute statement body if it is true */
      scanf("%d", &numbers[i]);
   }                                      /* do the update (i++) then jump back to LOOP */

   /* calculate the average */

   for (i = 0; i < SIZE; i++)
      sum += numbers[i];

   result = (float) sum / (float) SIZE;

   /* print out the text string the hard way, then display the result */

   i = 0;
   while (letters[i])
      printf("%c", letters[i++]);

   printf("%f\n", result);
}
```

```
thor> gcc -o arrayloop arrayloop.c
thor> arrayloop
Let's average 10 integers, shall we?

Enter integer #1: 27
Enter integer #2: 24
Enter integer #3: 28
Enter integer #4: 29
Enter integer #5: 21
Enter integer #6: 22
Enter integer #7: 22
Enter integer #8: 24
Enter integer #9: 27
Enter integer #10: 29
The average of the integers entered is 25.299999
thor>
```

```
/*
 *  lowercase.c  -  John K. Estell  -  22 April 1994
 *  read input (preferably redirected) from standard input,
 *  convert all upper case characters to lower case, then
 *  print out the results.
 */

#include <stdio.h>

main()
{
   char ch;

   /*
    *  while loop uses getchar() to read in one character from the
    *  standard input.  EOF is the pre-defined constant for
    *  End Of File - so as long as one doesn't read in the EOF
    *  value you're processing valid input.
    *  Please note that there are functions available for checking
    *  characters for certain properties in the ctype.h header file.
    */

   while ( (ch = getchar()) != EOF )   /* make assignment, then compare value against EOF */
   {
      if ((ch >= 'A') && (ch <= 'Z'))  /* check for uppercase character */
         ch += 'a' - 'A';              /* convert to lowercase the "hard way"... */
      putchar(ch);                     /* write character to standard output */
   }
}
thor> gcc -o lowercase lowercase.c


thor> cat text
THIS IS A TEST OF THE EMERGENCY BROADCAST SYSTEM
This is *only* a test.
thor> lowercase <text
this is a test of the emergency broadcast system
this is *only* a test.
```

In this program we had to use redirection of the standard input in order to process a file. Eventually we'll learn how to properly specify a file name on the command line and open it for processing.

```
/*
 *  lower2.c  -  John K. Estell  -  22 April 1994
 *  "Better" way of writing the lowercase.c program - uses the
 *  ctype.h functions.
 */

#include <stdio.h>
#include <ctype.h>

main()
{
   char ch;

   while ( (ch = getchar()) != EOF )
   {
      if (isupper(ch))     /* returns true if ch is upper case letter */
         ch = tolower(ch); /* converts upper case letter to lower case... */
      putchar(ch);
   }
}
```

```
/*
 *  lower3.c  -  John K. Estell  -  22 April 1994
 *  An "even better" way of writing the lowercase.c program - uses the
 *  ctype.h functions.
 */

#include <stdio.h>
#include <ctype.h>

main()
{
   char ch;

   while ( (ch = getchar()) != EOF )
      putchar(tolower(ch));  /* if ch is not upper case, function returns ch */
}
```

```
/*
 *  comma.c  -  John K. Estell  -  25 April 1994
 *  Demonstration of the comma operator through the exercise of averaging an input stream
 *  of floating point numbers.  Can also be considered an example of bad programming style.
 */

#include <stdio.h>

main()
{
   float num, sum = 0.0;
   int n = 0;

   /* The the user what to do... */

   printf("Let’s average a bunch of numbers\n");
   printf("Enter ’quit’ to terminate\n");

   /*
    *  Use comma expression within the while loop expression to prompt, then read, the
    *  data.  The scanf() function will return 1 if the input is floating-point, and
    *  will return either 0 or -1 is the input is not.  Assuming correct input...
    */

   while ( printf("Enter value #%d: ", n + 1) , scanf("%f", &num) == 1 )
   {
      sum += num;
      n++;
   }

   /* Display the result */

   printf("Average is %f\n", sum / (float) n );
}
```

```
thor> gcc -o comma comma.c
thor> comma
Let’s average a bunch of numbers
Enter ’quit’ to terminate
Enter value #1: 2.7
Enter value #2: 4.3
Enter value #3: 5.1
Enter value #4: 1.2
Enter value #5: -1
Enter value #6: 2.1
Enter value #7: 2.3
Enter value #8: quit
Average is 2.385714
thor>
```

A better solution would be to specify an infinite while loop, then test to see if scanf returns a value other than one. If so, then break out of the while loop.

character processing - frequency analysis

```c
/*
 *  frequency.c  -  John K. Estell  -  27 April 1994
 *  Determine the frequency that each letter of the alphabet
 *  appears in the input stream.
 */

#include <stdio.h>
#include <ctype.h>

main()
{
   unsigned long int num_letters = 0;
   unsigned long int letter[26];
   int      i;
   char     ch;

   /* explicitly initialize the array (don’t trust compilers to implicitly set values!) */

   for (i = 0; i < 26; i++)
      letter[i] = 0;

   /* read in characters */

   while ( (ch = getchar()) != EOF )
      if (isalpha(ch))         /* increase the counts if we have a letter */
      {
         num_letters++;        /* increase total letter count */
         if (islower(ch))      /* convert to upper case */
           ch = toupper(ch);
         letter[ch-’A’]++;     /* increase specific letter count by converting ch to */
                               /* proper array range of 0-25 */
      }

   /* print out the results */

   printf("\nTotal number of letters: %ld\n", num_letters);
   printf("\nFrequency:\n");

   for (i = 0; i < 26; i++)
      printf("’%c’: %6.3f%%\n", ’A’ + i,
             100.0 * (float) letter[i] / (float) num_letters);
}
```

```
thor> gcc -o frequency frequency.c
thor> ls -l ~cse406/tcsh.man
-rw-------  1 estell   misc       130144 Apr 27 16:27 tcsh.man
thor> frequency <tcsh.man

Total number of letters: 69856

Frequency:
'A':  7.046%
'B':  1.566%
'C':  4.695%
'D':  3.828%
'E': 11.843%
'F':  2.182%
'G':  1.655%
'H':  4.727%
'I':  7.162%
'J':  0.155%
'K':  0.600%
'L':  4.707%
'M':  3.255%
'N':  7.143%
'O':  6.967%
'P':  2.452%
'Q':  0.115%
'R':  5.809%
'S':  7.130%
'T':  9.502%
'U':  2.801%
'V':  0.908%
'W':  1.452%
'X':  0.693%
'Y':  1.509%
'Z':  0.099%
```

counting characters

```
/*
 *  charcount.c  -  John K. Estell  -  29 April 1994
 *  Simple demo of counting number of characters in an input stream
 */

#include <stdio.h>

main()
{
   unsigned long int characters = 0;

   while ( getchar() != EOF )
      characters++;

   printf("%d\n", characters);
}

thor> gcc -o charcount charcount.c
thor> ls -l ~cse406
total 288

-rwx--x--x  1 cse406   misc        49152 Mar 30  1994 pager*
-rw-r--r--  1 cse406   misc       130144 Nov  3 08:56 tcsh.man
-rw-r--r--  1 cse406   misc        28775 Mar 30  1994 tutor.vi

thor> wc -c ~cse406/tcsh.man     (using the Unix word count program to get character count)
  130144 /home/jupiter/cse406/tcsh.man
thor> charcount < ~cse406/tcsh.man   (our example should match the values given above)
130144
```

```
/*
 *  linecount.c  -  John K. Estell  -  29 April 1994
 *  Simple demo of counting number of lines in an input stream
 */

#include <stdio.h>

main()
{
   unsigned long int lines = 0;
   char     ch;

   while ( (ch = getchar()) != EOF )
      if (ch == '\n')
         lines++;

   printf("%d\n", lines);
}
```

```
thor> gcc -o linecount linecount.c
thor> wc -l ~cse406/tcsh.man
    3498 /home/jupiter/cse406/tcsh.man
thor> linecount < ~cse406/tcsh.man
3498
```

wrong way to count words

```
/*
 *  wordcount.c  -  John K. Estell  -  29 April 1994
 *  Simple demo of trying to count the number of words in an input stream
 *  by counting the number of whitespace characters.
 */

#include <stdio.h>

main()
{
   unsigned long int words = 0;
   char     ch;

   while ( (ch = getchar()) != EOF )
      if ( (ch == ' ') || (ch == '\t') || (ch == '\n') )
         words++;

   printf("%d\n", words);
}
```

```
thor> gcc -o wordcount wordcount.c
thor> wc -w ~cse406/tcsh.man
   15843 /home/jupiter/cse406/tcsh.man
thor> wordcount <~cse406/tcsh.man
45618
```

This program contains a logic error. The erroneous assumption is that whitespace characters can be counted to determine the number of words present. Unfortunately, this doesn't handle the cases where a sequence of whitespace characters are encountered....

```
/*
 *  wordcount1.c  -  John K. Estell  -  29 April 1994
 *  Better demo of trying to count the number of words in an input stream
 *  by keeping track of whether or not we’re within a word.
 */

#include <stdio.h>

#define NO  0
#define YES 1

main()
{
   unsigned long int words = 0;
   char     ch;
   int      in_word = NO;

   while ( (ch = getchar()) != EOF )
      if ( (ch == ’ ’) || (ch == ’\t’) || (ch == ’\n’) )
      {
         if (in_word)
         {
            words++;
            in_word = NO;
         }
      }
      else
         in_word = YES;  /* if it’s not whitespace then it’s a word... */


   printf("%d\n", words);
}

thor> gcc -o wordcount1 wordcount1.c
thor> wc -w ~cse406/tcsh.man
   15843 /home/jupiter/cse406/tcsh.man
thor> wordcount1 < ~cse406/tcsh.man
15843
```

The proper way to count the number of words is to count the number of non-whitespace character to whitespace character transitions in the data stream. To do this we must somehow examine all two-character sequences in the stream. Simple solution is to use a flag variable to keep track of whether or not we’re currently in a word. When the flag is set and whitespace is encountered then increment the word count and reset the flag. The flag will be set every time a non-whitespace character is encountered.

```
/*
 *  counter.c  -  John K. Estell  -  3 November 1994
 *  Primitive version of UNIX’s ’wc’ program.  Input is from
 *  standard input only in this version.  Output displays the
 *  number of lines, words, and characters in that order.
 *  Note that there are no extra lines or text in the output, for that would
 *  violate the philosophy of UNIX with respect to piping information
 *  to another program.
 */

#include <stdio.h>

#define YES 1
#define NO  0

main()
{
   unsigned long int characters = 0, words = 0, lines = 0;
   int in_word = NO;
   char ch;

   while ( ( ch = getchar() ) != EOF )
   {
      characters++;      /* increment character count */

      switch ( ch )
      {
         case ’\n’ : lines++;  /* fall through as newline requires end of word test */

         case ’\t’ :
         case ’ ’  : if ( in_word )
                     {
                        words++;
                        in_word = NO;
                     }
                     break;

         default :   in_word = YES;
      }
   }
   printf(" %7d %7d %7d\n", lines, words, characters);
}

thor> gcc -o counter counter.c
thor> wc ~cse406/tcsh.man
    3498   15843  130144 /home/jupiter/cse406/tcsh.man
thor> counter < ~cse406/tcsh.man
    3498   15843  130144
```

```
/*
 *  function0.c  -  John K. Estell  -  3 May 1994
 *  Trivial example of using a function in C.
 */

#include <stdio.h>


/*
 *  *** function prototypes - explained in next program ***
 */

void line_char(char ch, int length);


/*
 *  *** main ***
 */

main()
{
   int num;

   printf("Enter a positive integer: ");
   scanf("%d", &num);

   line_char('*', num);

   printf("The value of the integer is now %d.\n", num);
}

/*
 *  *** functions - all parameters are passed by value ***
 */

void line_char( char ch, int length )
{
   while (length--)
      putchar(ch);
   putchar('\n');
}
```

```
thor> gcc -o function0 function0.c
thor> function0


Enter a positive integer: 20
********************
The value of the integer is now 20.
```

formal demonstration of function use

```
/*
 *  function1.c  -  John K. Estell  -  3 May 1994
 *  Demonstration of function use
 */

#include <stdio.h>

/*
 *  function prototypes: this is where all functions are declared for later use in the
 *                       program.  This is done in order to define the type of what’s
 *                       returned by the function as well as the number and types of the
 *                       parameters passed to the function.  This allows the compiler to
 *                       protect the stack from improper parameter passing.
 */

int digit_sum( int n );  /* parameters only require data types, but should include the   */
                        /* variable name as well for documentation purposes             */
int get_integer( void ); /* "void" is used here to indicate that nothing is being passed */

/* main */

main()
{
   int x;

   printf("Digit sum - another exciting program\n");
   printf("   from Furball Enterprises, Inc.\n\n");
   x = get_integer();

   printf("Sum of the digits of %d is %d.\n", x, digit_sum(x) );  /* note use of function */
                                                               /* inside printf()      */
}

/* functions */

/*
 *  get_integer() - used to get an integer input from the user.
 *  This routine will not return until a valid integer is entered.
 */

int get_integer( void )
{
   int val;                 /* declare variables that are local to this function */
   int valid_input = 0;
   char inbuf[133];

   while (valid_input < 1)
   {
      printf("Enter an integer: ");
      gets(inbuf);
      valid_input = sscanf(inbuf, "%d", &val);
   }
   return val;  /* this returns an integer as the result of the function */
}
/*
 *  digit_sum() - will take the integer passed to it and return the sum
 *  of all of its digits.
 */

int digit_sum( int n )
{
   int sum = 0;

   if (n < 0)   /* assuming that n != INT_MIN, as -INT_MIN = INT_MIN in 2's complement */
               /* representation - different approach needed to handle this fencepost... */
      n = -n;   /* this algorithm requires a positive number */

   while (n != 0)
   {
      sum += n % 10;
      n /= 10;
   }

   return sum;
}

thor> gcc -o function1 function1.c
thor> function1
Digit sum - another exciting program
   from Furball Enterprises, Inc.

Enter an integer: 427
Sum of the digits of 427 is 13.
thor> function1
Digit sum - another exciting program
   from Furball Enterprises, Inc.

Enter an integer: -427
Sum of the digits of -427 is 13.
thor> function1
Digit sum - another exciting program
   from Furball Enterprises, Inc.

Enter an integer: 99999
Sum of the digits of 99999 is 45.
```

the infamous function-factorial program

```
/*
 *  function2.c  -  John K. Estell  -  3 May 1994
 *  Another infamous example of calculating the factorial of
 *  an integer number.
 */

#include <stdio.h>

/*
 *  function prototypes
 */

int factorial( int num );

/* main */

main()
{
   int n;

   printf("Enter an integer: ");
   scanf("%d", &n);

   printf("%d! = %d\n", n, factorial(n));
}

/* functions */

int factorial( int num )
{
   int i, result = 1;

   for (i = 2; i <= num; i++)
      result *= i;

   return result;
}
```

```
thor> gcc -o function2 function2.c
thor> function2
Enter an integer: 0
0! = 1
thor> function2
Enter an integer: 1
1! = 1
thor> function2
Enter an integer: 5
5! = 120
```

```
/*
 *  calculator.c  -  John K. Estell  -  2 May 1994
 *  Simple calculator program to demonstrate functions and
 *  parsing using scanf.
 *  Uses infix notation - input format is: operand1 operator operand2
 *  Compile with: gcc -o calculator calculator.c -lm
 */

#include <stdio.h>
#include <math.h>

/*** function prototypes ***/

double add(double x, double y);        /* addtion operation        */
double subtract(double x, double y);   /* subtraction operation    */
double multiply(double x, double y);   /* multiplication operation */
double divide(double x, double y);     /* division operation       */

/*** main program ***/

main()
{
   int valid_input = 1;    /* flag for keeping track of valid input */
   double x, y, result;    /* operands and result for operations    */
   char op;                /* operator                              */
   char inbuf[133];        /* input buffer                          */

   /* display program information */

   printf("Simple calculator - format is number operator number\n");
   printf("Supported operations:\n");
   printf("     + (add)  - (subt)  * (mult)  / (div)  ^ (power)\n\n");

   /*  obtain input from the user - "%lf" for double-precision floating point,
    *  " %c" to read next non-whitespace character.
    */

   printf("Enter equation: ");
   gets(inbuf);
   if (sscanf(inbuf, "%lf %c %lf", &x, &op, &y) != 3)
      valid_input = 0;

   /* process the operator and perform the indicated operation */

   switch (op)
   {
      case '+': result = add(x,y);      break;
      case '-': result = subtract(x,y); break;
      case '*': result = multiply(x,y); break;
      case '/': result = divide(x,y);   break;
      case '^': result = pow(x,y);      break;  /* using function in math.h */
      default : valid_input = 0;        break;
   }
   /* display the result */

   if (valid_input)
      printf("%lf %c %lf = %lf\n", x, op, y, result);
   else
      fprintf(stderr, "Error.\n"); /* writing error message to the standard error device */
}


/*** functions ***/

double add(double x, double y)
{
   return x + y;  /* evaluate the expression, then return the result */
}

double subtract(double x, double y)
{
   return x - y;
}

double multiply(double x, double y)
{
   return x * y;
}

double divide(double x, double y)
{
   return x / y;
}
```

```
thor> gcc -o calculator calculator.c -lm
thor> calculator
Simple calculator - format is number operator number
Supported operations:
     + (add)  - (subt)  * (mult)  / (div)  ^ (power)

Enter equation: 86 + 99
86.000000 + 99.000000 = 185.000000
thor> calculator
Simple calculator - format is number operator number
Supported operations:
     + (add)  - (subt)  * (mult)  / (div)  ^ (power)

Enter equation: 27.8 - 99.9
27.800000 - 99.900000 = -72.100000
thor> calculator
Simple calculator - format is number operator number
Supported operations:
     + (add)  - (subt)  * (mult)  / (div)  ^ (power)

Enter equation: 2*3
2.000000 * 3.000000 = 6.000000
thor> calculator
Simple calculator - format is number operator number
Supported operations:
     + (add)  - (subt)  * (mult)  / (div)  ^ (power)

Enter equation: 2 ^ 16
2.000000 ^ 16.000000 = 65536.000000
thor> calculator
Simple calculator - format is number operator number
Supported operations:
     + (add)  - (subt)  * (mult)  / (div)  ^ (power)

Enter equation: 2 ^ 1024
2.000000 ^ 1024.000000 = Inf
thor> calculator
Simple calculator - format is number operator number
Supported operations:
     + (add)  - (subt)  * (mult)  / (div)  ^ (power)

Enter equation: 0 / 0
0.000000 / 0.000000 = NaN
thor> calculator
Simple calculator - format is number operator number
Supported operations:
     + (add)  - (subt)  * (mult)  / (div)  ^ (power)

Enter equation: two + two
Error.
```

duration of a variable

```
/*
 *  duration.c  -  John K. Estell  -  8 May 1994
 *  Looks at duration of a variable
 */

#include <stdio.h>

/* global variables - to be declared outside of all functions */

int i = 0;

/* function prototypes */

int f( int x );
int g( int x );
int h( int x );

/* main function */

main()
{
   int k;

   printf("The initial value for i is %d.\n\n", i);

   for (k=0; k<5; k++)
      printf("f(%d) = %d\n", k, f(k));
   printf("\n");
   for (k=0; k<5; k++)
      printf("g(%d) = %d\n", k, g(k));
   printf("\n");
   for (k=0; k<5; k++)
      printf("h(%d) = %d\n", k, h(k));
   printf("\n");
   printf("The final value for i is %d.\n", i);
}

* functions */

/*
 *  f() - function using automatic storage duration: any value placed within
 *        variable will cease to exist once the function is exited.
 *        No initial value can be assumed.
 */

int f(int x)
{
   int j = 0;

   j += x;
   return j;
}
```

```
/*
 *  g() - function using globally defined variable as a method of static
 *        storage duration: any value placed within variable will exist as
 *        long as the program executes.  The global variable can be used
 *        within any function.
 */

int g(int x)
{

   i += x;
   return i;
}

/*
 *  h() - function explicitly using static storage duration: any value placed
 *        within variable will exist as long as the program executes, regardless
 *        of the number of times that the function is executed.  The initial
 *        value specified is valid only for the first time the function is
 *        executed.  The scope of the variable is limited to the function.
 */

int h(int x)
{
   static int j = 0; /* j = 0 only the first time function is used, then will vary... */

   j += x;
   return j;
}

thor> gcc -o duration duration.c
thor> duration
The initial value for i is 0.

f(0) = 0
f(1) = 1
f(2) = 2
f(3) = 3
f(4) = 4

g(0) = 0
g(1) = 1
g(2) = 3
g(3) = 6
g(4) = 10

h(0) = 0
h(1) = 1
h(2) = 3
h(3) = 6
h(4) = 10

The final value for i is 10.
```

```
/*
 *  function3.c  -  John K. Estell  -  9 May 1994
 *  Demonstration of using pointers with functions
 */

#include <stdio.h>

/* function prototypes */

int digit_sum( int n );
void get_integer( char *prompt, int *value );  /* must pass addresses as parameters */

/* main function */

main()
{
   int x;

   printf("Digit sum - another exciting program\n");
   printf("   from Furball Enterprises, Inc.\n");
   printf("*** The NEW and IMPROVED version! ***\n\n");

   get_integer("Enter an integer: ", &x);  /* literal string is treated as an address */

   printf("Sum of the digits of %d is %d.\n", x, digit_sum(x) );
}

/* functions */

/*
 *  get_integer() - used to get an integer input from the user.
 *  This routine will not return until a valid integer is entered.
 */

void get_integer( char *prompt, int *value )
{
   /* declare variables that are local to this function */

   int valid_input = 0;
   char inbuf[133];

   while (valid_input < 1)
   {
      printf(prompt);  /* name of array acts as pointer to first element of that array */
      gets(inbuf);
      valid_input = sscanf(inbuf, "%d", value); /* value is already an address... */
   }
}
```

```
/*
 *  digit_sum() - will take the integer passed to it and return the sum
 *  of all of its digits.
 */

int digit_sum( int n )
{
   int sum = 0;

   if (n < 0)   /* assume that n != INT_MIN */
      n = -n;   /* algorithm requires a positive number */

   while (n != 0)
   {
      sum += n % 10;
      n /= 10;
   }

   return sum;
}

thor> gcc -o function3 function3.c
thor> function3
Digit sum - another exciting program
   from Furball Enterprises, Inc.
*** The NEW and IMPROVED version! ***

Enter an integer: 42786
Sum of the digits of 42786 is 27.
```

```
/*
 *  ptrarray.c  -  John K. Estell  -  10 May 1994
 *  Illustration of pointer arithmetic when using arrays
 */

#include <stdio.h>
#define SIZE 6

main()
{
   int int_array[SIZE], *p_int, i;
   float float_array[SIZE], *p_float;
   char char_array[SIZE], *p_char;
   double double_array[SIZE], *p_double;

   /* initialize pointers */

   p_int = int_array;
   p_float = float_array;
   p_char = char_array;
   p_double = double_array;

   /* print out hexadecimal address locations of the array elements - include leading 0 */

   printf(" INDEX\t\t int_array\t float_array\t char_array\t double_array\n");
   for (i = 0; i < SIZE; i++)
      printf("p_type + %d: \t %0X \t %0X \t %0X \t %0X \n",
              i, p_int + i, p_float + i, p_char + i, p_double + i);

   /* again, print out address locations of the array elements */

   printf("\n");
   printf(" INDEX\t\t int_array\t float_array\t char_array\t double_array\n");
   for (i = 0; i < SIZE; i++)
      printf("p_type + %d: \t %0X \t %0X \t %0X \t %0X \n",
              i, p_int++, p_float++, p_char++, p_double++);
}
```

```
thor> gcc -o ptrarray ptrarray.c
thor> ptrarray
 INDEX          int_array       float_array     char_array      double_array
p_type + 0:     F7FFEC78        F7FFEC58        F7FFEC48        F7FFEC10
p_type + 1:     F7FFEC7C        F7FFEC5C        F7FFEC49        F7FFEC18
p_type + 2:     F7FFEC80        F7FFEC60        F7FFEC4A        F7FFEC20
p_type + 3:     F7FFEC84        F7FFEC64        F7FFEC4B        F7FFEC28
p_type + 4:     F7FFEC88        F7FFEC68        F7FFEC4C        F7FFEC30
p_type + 5:     F7FFEC8C        F7FFEC6C        F7FFEC4D        F7FFEC38

 INDEX          int_array       float_array     char_array      double_array
p_type + 0:     F7FFEC78        F7FFEC58        F7FFEC48        F7FFEC10
p_type + 1:     F7FFEC7C        F7FFEC5C        F7FFEC49        F7FFEC18
p_type + 2:     F7FFEC80        F7FFEC60        F7FFEC4A        F7FFEC20
p_type + 3:     F7FFEC84        F7FFEC64        F7FFEC4B        F7FFEC28
p_type + 4:     F7FFEC88        F7FFEC68        F7FFEC4C        F7FFEC30
p_type + 5:     F7FFEC8C        F7FFEC6C        F7FFEC4D        F7FFEC38
```

Note that on this particular machine, the difference between successive integer array elements is 4 bytes; for floating point elements, 4 bytes; for character array elements, 1 byte; and for double-precision floating point elements, 8 bytes. Also note the alignment of the addresses: the data types that are 4 bytes in size have elements that start on addresses that are divisible by 4, and the double type elements start on addresses divisible by 8. This alignment is required for efficient use of the processor as all the bytes for a particular datum can be accessed in a minimum number of memory reads or writes.

an example of a bus error

```c
/*
 *  buserror.c  -  John K. Estell  -  5 May 1995
 *  THIS PROGRAM HAS A DELIBERATE ERROR!!!
 *
 *  This program will illustrate a condition that will cause
 *  a "bus error" when attempting to execute the program.
 *  The program will attempt to read in ten integers and
 *  calculate the average.
 */

#include <stdio.h>
#define SIZE 10

main()
{
   int sum = 0;
   int valid_input;
   char inbuf[133];
   char values[SIZE];
   int i;

   for (i = 0; i < SIZE; i++)
   {
      valid_input = 0;

      while (!valid_input)
      {
         printf("Enter integer #%d: ", i + 1);
         gets(inbuf);

         valid_input = sscanf(inbuf, "%d", &values[i]);

         switch (valid_input)
         {
            /* -1 is returned when inbuf consists only of whitespace */

            case -1:
               printf("Please enter an integer.\n\n");
               valid_input = 0;   /* as -1 is interpreted as true... */
               break;

            /* 0 is returned if the first entry in inbuf is not int */

            case 0:
               printf("That was not an integer - please try again.\n\n");
         }
      }
   }

   for (i = 0; i < SIZE; i++)
      sum += values[i];

   printf("Average is %.1f.\n", (float) sum / (float) SIZE);
}
```

```
thor> gcc -o buserror buserror.c
thor> buserror
Enter integer #1:
Please enter an integer.

Enter integer #1: an integer
That was not an integer - please try again.

Enter integer #1: 46
Enter integer #2: 27
Bus error (core dumped)
thor> exit
```

Let's modify the buserror program to print out the address where the read in value is to be stored:

```
        :
        :
       while (!valid_input)
       {
          int a;
          printf("Enter integer #%d: ", i + 1);
          gets(inbuf);

          printf("i=%d &values = %p &values[i] = %p\n", i, values, values + i);

          valid_input = sscanf(inbuf, "%d", &values[i]);

          switch (valid_input)
            :
            :
```

```
thor> buserror
Enter integer #1: 46
i=0 &values = f7ffe9b8 &values[i] = f7ffe9b8
Enter integer #2: 27
i=1 &values = f7ffe9b8 &values[i] = f7ffe9b9 (Note that this address is not divisible by 4)
Bus error (core dumped)
thor> exit
```

Now we can see the problem! On this (and many other) systems, integers have to be stored on a "longword" boundary - in simple terms, the address has to be divisible by 4. (To know the reason why this is so you'd have to know something about computer architecture.) In line 19 we accidently declared our values array to be of type char instead of type int. The first store was performed as we were aligned on a longword. The second store, however, caused an error as the address value was incremented by one (which is appropriate for a character array) instead of by four (which is appropriate for an integer array). Therefore, when you encounter a bus error, it means that your program tried to save a value to a memory location that is not properly aligned for the given data type.

```
/*
 *  funarray.c  -  John K. Estell  -  9 May 1994
 *  Demonstration of using pointers with arrays
 */

#include <stdio.h>

#define SIZE 6

/* function prototypes */

void   get_integer( int *array, int index ); /* passing starting address and offset */
double average_values( int *array, int array_size );

main()
{
   int i, table[SIZE];

   printf("It’s the \"Average %d Integer Numbers\" program!!!\n", SIZE);

   for (i = 0; i < SIZE; i++)
      get_integer(table, i);

   printf("Average is %.4f\n", average_values(table, SIZE));
}

/* functions */

void get_integer( int *array, int index )
{
   int valid_input = 0;
   char inbuf[133];

   while (valid_input < 1)
   {
      printf("Enter integer #%d: ", index + 1);
      gets(inbuf);
      valid_input = sscanf(inbuf, "%d", &array[index]);
   }
}

double average_values( int *array, int array_size )
{
   double sum = 0.0;
   int i = 0;

   while (i < array_size)
      sum += *(array + i++);

   return (sum / (double) array_size);
}
```

```
thor> gcc -o funarray funarray.c
thor> funarray
It’s the "Average 6 Integer Numbers" program!!!
Enter integer #1: 4
Enter integer #2: 5
Enter integer #3: a tisket, a tasket, a furball in a basket
Enter integer #3: 6
Enter integer #4: 7
Enter integer #5: 8
Enter integer #6: 9
Average is 6.5000
```

This program references array elements using two similar methods. The "traditional" method, shown in the get_integer function, is array_name[index], where array_name constitutes a base address and index provides an offset into the array that refers to the desired element. To access the element, the system must calculate the address of the element by taking the base address and adding to it the value of the index times the size in bytes of an array element.

The other method used is shown in the average_values function. Here we are explicitly showing the array address calculations by using the dereferencing operator on the address formed by adding the integer index to the base address. The act of adding an integer to an address will cause the system to multiply the integer value by the size of the data being reference by the base pointer prior to the addition being performed.

A faster method is to explicitly use pointers; an example would be:

```
int *ptr, i;
ptr = array;
for (i = 0; i < array_size; i++)
   sum += *(ptr++);
```

This removes the need for multiplying by the data size, as the incrementing of a pointer variable is always by the size of the data being referred to. This concept concerning address arithmetic is why we make a distinction between pointers to different types, as it allows the system to know what the difference in bytes is between two consecutive elements.

```
/*
 *  dynamic.c  -  John K. Estell  -  16 May 1994
 *  Demo of a function using a pointer to a dynamically allocated array
 */

#include <stdio.h>
#include <malloc.h>

/* function prototypes */

double average_numbers(int *array, int n);

main()
{
   int *table, i, n;

   printf("Average some integer numbers\n");
   printf("How many integers? ");
   scanf("%d", &n);

   /*
    *  malloc allocates bytes - to get enough space for storing
    *  data one must multiply number of items by the size of
    *  said item, in this case n * sizeof(int).
    *  Use a cast to properly set the type for the output of
    *  malloc from "pointer to void" to "pointer to int", then check
    *  to make sure that you obtained the requested memory.
    */

   table = (int *) malloc( n * sizeof(int) );

   if (table == NULL)
   {
      fprintf(stderr, "Insufficient memory.\n");
      exit(1);   /* exit the program on fatal error - return non-zero value on failure */
   }

   /*
    *  table now points to the first element in the dynamically
    *  allocated integer array.
    */

   for (i=0; i<n; i++)
   {
      printf("Enter integer #%d: ", i+1);
      scanf("%d", &table[i]);
   }
   printf("Average is %.12f\n", average_numbers( table, n ));
}
```

```
/*
 *  average_numbers - returns the average of a list of integer elements.
 *    input: address of array, size of array
 *  output: average of the elements in the array
 */


double average_numbers( int *array, int n )
{
   double sum = 0.0;
   int *p;

   p = array;
   while (p < array + n)
      sum += (double) *p++;

   return sum / (double) n;
}
```

```
thor> gcc -o dynamic dynamic.c
thor> dynamic
Average some integer numbers
How many integers? 7
Enter integer #1: 9
Enter integer #2: 8
Enter integer #3: 7
Enter integer #4: 9
Enter integer #5: 8
Enter integer #6: 7
Enter integer #7: 9
Average is 8.142857142857
thor> dynamic
Average some integer numbers
How many integers? 10
Enter integer #1: 427
Enter integer #2: 4789
Enter integer #3: 4763
Enter integer #4: 2891
Enter integer #5: 47893
Enter integer #6: 389
Enter integer #7: 3891
Enter integer #8: 4768
Enter integer #9: 2378
Enter integer #10: 34
Average is 7222.300000000000
```

dynamic allocation of two-dimensional arrays

```
/*
 *  twod.c  -  John K. Estell  -  2 June 1995
 *  Example of how to declare and use a two-dimensional array that can be
 *  passed to a function.  For a truly dynamic implementation, replace ROW and
 *  COL with variables...
 */

#include <stdio.h>

#define ROW 10
#define COL 12

int sum(int *a[]);

main()
{
   int **array;  /* declare an array of pointers for the rows */
   int i,j;

   array = (int **) malloc(ROW * sizeof(int *));

   /* for each pointer dynamically allocate a row having correct number of columns.  */

   for (i = 0; i < ROW; i++)
     array[i] = (int *) malloc(COL * sizeof(int)); /* point to allocated storage for row */

   /* initialize the array */

   for (i = 0; i < ROW; i++)
      for (j = 0; j < COL; j++)
         array[i][j] = i*j;     /* note that we can access elements normally */

   printf("sum = %d\n", sum(array));  /* here we pass the array */
}

/* sum: sums the contents of the array passed to it. */

int sum(int *a[])
{
   int i, j;
   int sum = 0;

   for (i = 0; i < ROW; i++)
   {
      for (j = 0; j < COL; j++)
      {
         sum += a[i][j];
         printf("%4d", a[i][j]);  /* added feature - for illustrative purposes only */
      }
      printf("\n");               /* added feature - for illustrative purposes only */
   }
   return sum;
}
```

```
thor> twod
   0   0   0   0   0   0   0   0   0   0   0   0
   0   1   2   3   4   5   6   7   8   9  10  11
   0   2   4   6   8  10  12  14  16  18  20  22
   0   3   6   9  12  15  18  21  24  27  30  33
   0   4   8  12  16  20  24  28  32  36  40  44
   0   5  10  15  20  25  30  35  40  45  50  55
   0   6  12  18  24  30  36  42  48  54  60  66
   0   7  14  21  28  35  42  49  56  63  70  77
   0   8  16  24  32  40  48  56  64  72  80  88
   0   9  18  27  36  45  54  63  72  81  90  99
sum = 2970
```

```
/*
 *  flags.c  -  John K. Estell  -  17 May 1994
 *  wordcount program that uses command line arguments for determining output parameters.
 *  All passed parameters must be flags; input is still from standard input.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define YES 1
#define NO  0

/* function prototypes */

void report_fatal_error(char *pgm_name, char *param, char *error_msg);

/* main function: argc = number of arguments passed, argv = array of argument strings */

int main(int argc, char *argv[])
{
   int characters = 0, words = 0, lines = 0,
       in_word = NO,
       i, j;
   int cflag = YES, wflag = YES, lflag = YES; /* default settings */
   char ch;

   /* check to see if any arguments were passed */

   if (argc > 1)
   {
      cflag = NO;  /* if arguments have been passed then enable only */
      wflag = NO;  /* those options that have been selected          */
      lflag = NO;

      /*
       *  go through passed arguments - check first character of each argument to
       *  see if it is a flag argument.
       */

      for (i = 1; i < argc; i++)
         if (argv[i][0] != '-')  /* if first character of i-th entry is not a dash... */
            report_fatal_error(argv[0], argv[i], "is not a flag");
         else
            for (j = 1; j < strlen(argv[i]); j++)  /* process the flag */
               switch( argv[i][j] )
               {
                  case 'l' : lflag = YES; break;
                  case 'w' : wflag = YES; break;
                  case 'c' : cflag = YES; break;
                  default  : report_fatal_error(argv[0], argv[i],
                                                "is not a legal flag");
               }
   }
   /* process the characters... */

   while ( ( ch = getchar() ) != EOF )
   {
      characters++;      /* increment character count */
      switch ( ch )
      {
         case '\n' : lines++;          /* on newline increment line count */
         case '\t' :
         case ' '  : if ( in_word == YES )
                     {
                        words++;       /* if here have whitespace */
                        in_word = NO;
                     }
                     break;
         default :   in_word = YES;    /* have non-whitespace character */
      }
   }

   if (lflag) printf("%8d", lines);
   if (wflag) printf("%8d", words);
   if (cflag) printf("%8d", characters);
   printf("\n");
   exit(0); /* exit program - successful execution */
}

/* functions */

void report_fatal_error(char *pgm_name, char *param, char *error_msg)
{
   fprintf(stderr, "%s: ERROR - \"%s\" %s\n", pgm_name, param, error_msg);
   fprintf(stderr, "Usage is: %s [-l] [-w] [-c] <textfile\n", pgm_name);
   exit(1);  /* exit program - unsuccessful completion */
}
```

```
thor> gcc -o flags flags.c
thor> wc <flags.c
      87     350    2474
thor> flags <flags.c
      87     350    2474
thor> flags -l <flags.c
      87
thor> flags -w <flags.c
     350
thor> flags -c <flags.c
    2474
thor> flags -l -c <flags.c
      87    2474
thor> flags -lc <flags.c
      87    2474
thor> flags -l -badflag -c <flags.c
flags: ERROR - "-badflag" is not a legal flag
Usage is: flags [-l] [-w] [-c] <textfile
thor> flags badflag <flags.c
flags: ERROR - "badflag" is not a flag
Usage is: flags [-l] [-w] [-c] <textfile
thor> mv flags wordcount
thor> wordcount -l -w -c <flags.c
      87     350    2474
thor> wordcount -badflag <flags.c
wordcount: ERROR - "-badflag" is not a legal flag
Usage is: wordcount [-l] [-w] [-c] <textfile
```

```
/*
 *  copy.c  -  John K. Estell  -  23 May 1994
 *  Demonstration of file handling
 */

#include <stdio.h>
#include <stdlib.h>
#define BUFFER_SIZE 512

int main(int argc, char *argv[])
{
   char buffer[BUFFER_SIZE];
   FILE *fpin, *fpout;

   if (argc != 3)       /* check number of arguments */
   {
      fprintf(stderr, "Usage: %s sourcefile destinationfile\n", argv[0]);
      exit(1);
   }

   /* open files */

   fpin = fopen(argv[1], "r"); /* open filename specified in command line for reading */
   if (fpin == NULL)           /* fopen returns NULL upon error in opening the file */
   {
      fprintf(stderr, "%s: can’t open %s for input\n", argv[0], argv[1]);
      exit(1);
   }
   fpout = fopen(argv[2], "w"); /* open filename specified in command line for writing */
   if (fpout == NULL)
   {
      fprintf(stderr, "%s: can’t open %s for output\n", argv[0], argv[2]);
      exit(1);
   }

   /* copy the input file to the source file */

   while ( fgets(buffer, sizeof(buffer), fpin) != NULL ) /* haven’t reached end of file */
      fputs(buffer, fpout);

   /* close files */

   if (fclose(fpin) != 0)
   {
      fprintf(stderr, "%s: unable to close %s\n", argv[0], argv[1]);
      exit(1);
   }
   if (fclose(fpout) != 0)
   {
      fprintf(stderr, "%s: unable to close %s\n", argv[0], argv[2]);
      exit(1);
   }
   return 0;
}
```

```
thor> gcc -o copy copy.c
thor> cat text
THIS IS A TEST OF THE EMERGENCY COMPUTER SHUTDOWN SYSTEM
This is *only* a test.
If this was an actual emergency, you would be looking
at a blank screen wondering where your program has
disappeared to!!!
thor> copy text text.a
thor> cat text.a
THIS IS A TEST OF THE EMERGENCY COMPUTER SHUTDOWN SYSTEM
This is *only* a test.
If this was an actual emergency, you would be looking
at a blank screen wondering where your program has
disappeared to!!!
thor> diff text text.a
thor> copy badfile text.b
copy: can’t open badfile for input
thor> copy
Usage: copy sourcefile destinationfile
thor> copy badfile
Usage: copy sourcefile destinationfile
```

```
/*
 *  files.c  -  John K. Estell  -  23 May 1994
 *  version of wordcount program using a filelist but no flags
 */

#include <stdio.h>

#define YES 1
#define NO  0

main(int argc, char *argv[])
{
   int index, inword;
   int ch;
   FILE *fp;
   unsigned long lines, words, characters;
   unsigned long tlines = 0, twords = 0, tcharacters = 0;

   /* set defaults */

   index = 1;   /* argument - file number pointer */
   fp = stdin;  /* assume input is from standard input unless otherwise noted */

   while (index != argc)
   {
      if (argc > 1)
         if (fp=fopen(argv[index], "r")) == NULL)
         {
            fprintf(stderr, "%s: %s: No such file or directory\n",
                    argv[0], argv[index++]);
           continue; /* if we can’t open one file it’s not a fatal error for the program */
         }

      /* file exists, so process it... */

      lines = words = characters = 0;
      inword = NO;                       /* zero counters and flag */

      while ((ch=getc(fp)) != EOF)   /* process the current file */
      {
         characters++;               /* increment character count */
         if (ch == ’\n’)
            lines++;                 /* increment line count */
         if ((ch == ’ ’) || (ch == ’\t’) || (ch == ’\n’))
            inword = NO;
         else if (inword == NO)      /* increment word count */
         {
            inword = YES;
            words++;
         }
      }
      /* output result for the current file */

      printf(" %7d %7d %7d", lines, words, characters);

      /* print out filename - ignore if input stream is from standard input */

      if (argc > 1)
         printf(" %s\n", argv[index]);
      else
         printf("\n");

      /* close current file and update total counts */

      fclose(fp);

      tlines += lines;
      twords += words;
      tcharacters += characters;
      index++;
   }

   /* if there was more than one file, display total for all files */

   if (argc > 2)
      printf(" %7d %7d %7d total\n", tlines, twords, tcharacters);

   exit(0); /* exit program - return successful execution */
}

thor> gcc -o files files.c
thor> wc files.c flags.c
      85     307    2077 files.c
      87     350    2474 flags.c
     172     657    4551 total
thor> files files.c flags.c
      85     307    2077 files.c
      87     350    2474 flags.c
     172     657    4551 total
thor> wc files.c badfilename flags.c
      85     307    2077 files.c
wc: badfilename: No such file or directory
      87     350    2474 flags.c
     172     657    4551 total
thor> files files.c badfilename flags.c
      85     307    2077 files.c
files: badfilename: No such file or directory
      87     350    2474 flags.c
     172     657    4551 total
thor> cat files.c flags.c | files
     172     657    4551
```

```
/*
 *  wordcount.c  -  John K. Estell  -  23 May 1994
 *  UNIX wc "workalike" program, complete with files and flags.
 */

#include <stdio.h>
#include <stdlib.h>

#define YES 1
#define NO  0
#define UsageMesg "Usage: %s [-lwc] [filelist]\n"
#define OpenMesg  "%s: %s: No such file or directory\n"

main(int argc, char *argv[])
{
   int characters, words, lines;                 /* count per file */
   int tcharacters = 0, twords = 0, tlines = 0;  /* count for all files */
   int ch, in_word, index, j, fileptr;
   int cflag = YES, wflag = YES, lflag = YES;    /* default settings */
   FILE *fp;

   fp = stdin;         /* default input */
   fileptr = argc;     /* assuming no filenames as arguments */

   /* *** check for passed arguments - assume all flags appear before filelist *** */

   if (argc > 1)
   {
      if (argv[1][0] == '-') /* any flags? */
      {
         cflag = NO;  /* if flag arguments have been passed then enable only */
         wflag = NO;  /* those options that have been selected               */
         lflag = NO;
      }
      for (index = 1; index < argc; index++)    /* go through passed arguments */
         if (argv[index][0] == '-')  /* look to see if it is a flag */
         {
            for (j = 1; j < strlen(argv[index]); j++)  /* process the flag */
               switch( argv[index][j] )
               {
                  case 'l' : lflag = YES; break; /* output line count */
                  case 'w' : wflag = YES; break; /* output word count */
                  case 'c' : cflag = YES; break; /* output character count */
                  default  : fprintf(stderr, UsageMesg, argv[0]);
                             exit(1);
               }
          }
          else
          {
             fileptr = index;
             break;        /* remaining arguments are filenames */
          }
   }
   while (index < argc) /* perform actual counts of lines, words, and characters */
   {
      /*
       *  if file(s) have been specified, check to see if they're
       *  available and open the file if it is there.
       *  SKIP THIS if stdin is being used!
       */

      if ( fileptr != argc )
         if ( fp = fopen(argv[index], "r") ) == NULL )
         {
            fprintf(stderr, OpenMesg, argv[0], argv[index++]);   /* can't read file */
            continue;
         }

      characters = words = lines = 0;  /* zero counters and flag */
      in_word = NO;

      while ( ( ch = getc(fp) ) != EOF ) /* process the current file */
      {
         characters++;      /* increment character count */

         if ( ch == '\n')   /* increment line count */
            lines++;

         if ((ch == ' ') || (ch == '\t') || (ch == '\n')) /* have whitespace */
            in_word = NO;
         else if (in_word == NO) /* increment word count */
         {
            in_word = YES;
            words++;
         }
      }

      /* output current file results - print only what was specified */

      if (lflag) printf(" %7ld", lines);
      if (wflag) printf(" %7ld", words);
      if (cflag) printf(" %7ld", characters);

      if (argc != fileptr)
         printf(" %s\n", argv[index]); /* print filename */
      else
         printf("\n");

      /* clean up and do updates */

      fclose(fp);                  /* close current input file */
      tcharacters += characters;  /* update total counts for all files */
      twords += words;
      tlines += lines;
      index++;
   }
```

```
   /* *** output totals if more than one file was processed *** */

  if (argc - fileptr > 1) /* then have two or more files, so print out total count(s) */
  {
    if (lflag) printf(" %7ld", tlines);
    if (wflag) printf(" %7ld", twords);
    if (cflag) printf(" %7ld", tcharacters);
    printf(" total\n");
  }
  exit(0);  /* exit program - return status normal: successful execution */
}
```

Note that, while this is a workalike program, this is not the most efficient implementation possible for this code. For example, the character count can be more readily obtained by using an advanced system function that will return the attributes for a specified file. Things that are not being requested by the user could be skipped, thereby saving additional time.

rotate 13 encryption

```
/*
 *  rot13.c  -  John K. Estell  -  17 February 1993
 *  Performs rot13 encoding/decoding of a file
 *  input from either file or stdin - output to stdout
 */

#include <stdio.h>
#include <stdlib.h>

main( int argc, char *argv[] )
{
  int ch, chlower;
  FILE *fpin;

  switch (argc)    /* check number of arguments */
  {
    /* one argument - file not specified, so input from stdin */

    case 1: fpin = stdin;
            break;

    /* two arguments - open specified file and check to see if it exists */

    case 2: fpin = fopen(argv[1], "r");
            if (fpin == NULL)
            {
              fprintf(stderr, "%s: file not found\n", argv[1]);
              exit(1);
            }
            break;

    /* three or more arguments - report error condition */

    default:
            fprintf(stderr, "%s: too many arguments\n", argv[0]);
            exit(1);
  }

  /* read in the characters in the file */

  while ( (ch = getc(fpin)) != EOF )
  {
    chlower = ch | 0x20;  /* fast convert to lower case for testing purposes - can get */
                          /* away with this as the original value is unchanged...      */
    if ((chlower >= 'a') && (chlower <= 'm'))  /* so now only 2 comparisons are needed */
      ch += 13;
    if ((chlower >= 'n') && (chlower <= 'z'))
      ch -= 13;
    putc(ch, stdout);   /* print out the character to the standard output */
  }

  fclose(fpin);
  exit(0);
}
```

```
thor> gcc -o rot13 rot13.c
thor> cat text
The quick brown fox jumped over the lazy dog.
Hello!  This is a test!  This is only a test!
01234567890
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
!@#$%^&*()
That's all folks!!!
thor> rot13 a b
rot13: too many arguments
thor> rot13 a
a: file not found
thor> rot13 text
Gur dhvpx oebja sbk whzcrq bire gur ynml qbt.
Uryyb!  Guvf vf n grfg!  Guvf vf bayl n grfg!
01234567890
NOPQRSTUVWXYZABCDEFGHIJKLM
nopqrstuvwxyzabcdefghijklm
!@#$%^&*()
Gung'f nyy sbyxf!!!
thor> rot13 text >text.out
thor> rot13 text.out
The quick brown fox jumped over the lazy dog.
Hello!  This is a test!  This is only a test!
01234567890
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
!@#$%^&*()
That's all folks!!!
thor> cat text.out | rot13
The quick brown fox jumped over the lazy dog.
Hello!  This is a test!  This is only a test!
01234567890
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
!@#$%^&*()
That's all folks!!!
```

```
/*
 *  randomaccess.c  -  John K. Estell  -  2 December 1994
 *  Check to see exactly where fseek goes to for end of file
 */

#include <stdio.h>
#ifndef SEEK_SET
#define  SEEK_SET 0
#define  SEEK_END 2
#endif

main(int argc, char *argv[])
{
   char ch;
   long int location;
   FILE *fp;

   if ( (fp = fopen(argv[1], "r")) == NULL)
   {
      fprintf(stderr, "Error opening file: %s\n", argv[1]);
      exit(1);
   }

   /*
    *  go to the end of file and get its location - in this case ftell will return
    *  the total number of actual characters in the file; another interpretation is
    *  the byte location of the end of file in the data stream.
    */

   fseek(fp, 0, SEEK_END);
   location = ftell(fp);

   printf("ftell returns %d for ending position\n", location);

   /* print out the contents of the file in reverse byte order */

   while (location >= 0)
   {
      fseek(fp, location, SEEK_SET); /* set to point to byte at specified location */
      ch = getc(fp);
      printf("char #%3d = 0x%02X", location, ch); /* print hex value of character */
      if ( isprint(ch) )
         printf(" '%c'", ch);
      printf("\n");
      location--;
   }

   fclose(fp);
   exit(0);
}
```

```
thor> gcc -o randomaccess randomaccess.c
thor> cat text
ABCDE
FGHIJ
thor> wc text
       2       2      12 text
thor> randomaccess text
ftell returns 12 for ending position   (meaning there are 12 actual characters...)
char # 12 = 0xFFFFFFFF             (the 13th character, in location 12, is the end of file)
char # 11 = 0x0A                    (and the actual characters are in locations 0-11)
char # 10 = 0x4A 'J'
char #  9 = 0x49 'I'
char #  8 = 0x48 'H'
char #  7 = 0x47 'G'
char #  6 = 0x46 'F'
char #  5 = 0x0A
char #  4 = 0x45 'E'
char #  3 = 0x44 'D'
char #  2 = 0x43 'C'
char #  1 = 0x42 'B'
char #  0 = 0x41 'A'
thor>
```

For going to a specific location in a file, it is best to use SEEK_SET (the beginning of the file) as your base, and use the actual character position you want to go to as your offset. The beginning of a file remains constant whereas the end of the file can change (such as with an append operation) and trying to handle relative offsets with SEEK_CUR involves too many headaches.

```
/*
```

the dangers of the feof function

```
/*
 *  feof.c  -  John K. Estell  -  23 September 1995
 *  Example as to why one should not use the feof function with gleeful abandon.
 *  The problem is very subtle: feof will not return a non-zero
 *  to indicate end-of-file until an attempt is made to read
 *  past the end-of-file.  This will result in an extra "read"
 *  of the last line if you believe that C code is just like Pascal....
 */

#include <stdio.h>

main(int argc, char *argv[])
{
   int position;
   char buffer[133];
   FILE *fp;
   char *i;

   if (argc == 2)
      fp = fopen(argv[1], "r");
   else
      exit(1);

   if (fp == NULL)
      exit(1);

   while (!feof(fp))   /* this is a stupid thing to do in C! */
   {
      i = fgets(buffer, sizeof(buffer), fp);
      printf("%p: ", i);
      fputs(buffer, stdout);
   }

   fclose(fp);
}
```

```
thor> gcc -o feof feof.c
thor> cat text
To quote Laurie Sue:
  "To err is human,
   to moo, bovine."
thor> feof text
f7ffe9b0: To quote Laurie Sue:
f7ffe9b0:   "To err is human,
f7ffe9b0:    to moo, bovine."
0:    to moo, bovine."              (and here's our error!)
thor>
```

```
/*
 *  fgetsdemo.c  -  John K. Estell  -  16 June 1996
 *  Proper use of the feof and ferror functions.
 *  Key thing to note is that fgets returns NULL when it can no
 *  longer read in input; however, the function doesn't indicate
 *  why this is so.  feof and ferror are to be used only after
 *  fgets returns NULL in order to determine why the value was returned.
 */

#include <stdio.h>

int main(int argc, char **argv)
{
   char buffer[BUFSIZ];
   FILE *fp;

   if (argc != 2)   /* then have wrong number of arguments */
   {
      fprintf(stderr, "Usage: %s filename\n", argv[0]);
      exit(1);
   }

   if (( fp = fopen(argv[1], "r") ) == NULL)   /* can't open file for input */
   {
      perror(argv[1]);
      exit(2);
   }

   while (1)   /* process the input file */
   {
      if (fgets(buffer, sizeof(buffer), fp) == NULL)   /* no more input... */
      {
         if (feof(fp))     /* ... because end of file was reached */
            printf("Encountered end of file\n");
         if (ferror(fp))   /* ... because error was encountered while reading */
            printf("Encountered error\n");
         fclose(fp);
         break;
      }
      fputs(buffer, stdout);
   }
}
```

```
thor> fgetsdemo
Usage: fgetsdemo filename
thor> fgetsdemo badfilename
badfilename: No such file or directory
thor> fgetsdemo text
To quote Laurie Sue:
  "To err is human,
   to moo, bovine."
Encountered end of file
thor>
```

```
/*
 *  wagtail.c  -  John K. Estell  -  24 May 1994
 *  Simple version of UNIX tail command: given a file, it’ll display
 *  the last ten lines of the file, or the whole file if ten lines or fewer.
 */

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>

/*
 *  define statements needed as stdio.h doesn’t always contain the following
 *  symbolic constants (SVR4 did, BSD4.3 did not when this was written).
 */

#ifndef SEEK_SET
#define SEEK_SET 0L
#define SEEK_CUR 1L
#define SEEK_END 2L
#endif

/* main function */

int main( int argc, char *argv[] )
{
   char ch;
   FILE *fp;
   long int start, location;
   int  count = -1;

   if (argc != 2) /* then have improper number of command line arguments */
   {
      fprintf(stderr, "Usage: %s textfilename\n", argv[0]);
      exit(1);
   }

   /* open the specified file */

   fp = fopen(argv[1], "r");
   if (fp == NULL)
   {
      fprintf(stderr, "%s: error in opening \"%s\"\n", argv[0], argv[1]);
      perror(argv[0]);
      exit(1);
   }

   /* find the starting and ending character locations in the file */

   start = ftell(fp);
   fseek(fp, 0, SEEK_END);
   location = ftell(fp);
   /* find the tenth line from the end by working backwards through the file */

   while ( (count < 10) && (location > start) )
   {
      fseek(fp, location, SEEK_SET);
      if (getc(fp) == '\n')
         count++;
      location--;
   }

   /* print out the last lines in the file */

   if (location != start)
      location += 2;      /* need to adjust location pointer accordingly... */

   fseek(fp, location, SEEK_SET);

   while ( (ch = getc(fp)) != EOF )
      putchar(ch);

   /* clean up and exit program */

   fclose(fp);
   exit(0);
}

thor> gcc -o wagtail wagtail.c
thor> cat -n wagtail.c > wagtail.num
thor> wagtail wagtail.num
    69
    70     while ( (ch = getc(fp)) != EOF )
    71        putchar(ch);
    72
    73     /* clean up and exit program */
    74
    75     fclose(fp);
    76     exit(0);
    77  }
    78
thor> wc text
       5      36     203 text
thor> wagtail text
THIS IS A TEST OF THE EMERGENCY COMPUTER SHUTDOWN SYSTEM
This is *only* a test.
If this was an actual emergency, you would be looking
at a blank screen wondering where your program has
disappeared to!!!
thor> wagtail badfilename
wagtail: error in opening "badfilename"
wagtail: No such file or directory
thor> wagtail file1 file2
Usage: wagtail textfilename
thor> wagtail
Usage: wagtail textfilename
```

```
/*
 *  structures.c  -  John K. Estell  -  24 May 1994
 *  Example of dynamic structure allocation
 */

#include <stdio.h>

#define NAME_SIZE 40
#define BUF_SIZE  30

/* typedef allows you to define your own data type for later convenience. */

typedef
   struct
   {
      char name[NAME_SIZE];
      int id;
   } student;

/* function prototype */

student getdata( int i );  /* note that we can have a function return a structure... */

/* main function */

main()
{
   student *cse406;        /* type is "pointer-to-student" */
   int i, num_students;
   char buffer[BUF_SIZE];

   /* get the number of students */

   printf("Number of students: ");
   fgets(buffer, sizeof(buffer), stdin);
   sscanf(buffer, "%d", &num_students);

   /* allocate space for the student records */

   cse406 = (student *) malloc( num_students * sizeof(student) );

   /* get student information */

   for (i = 0; i < num_students; i++)
      cse406[i] = getdata(i);

   /* print out student information */

   for (i = 0; i < num_students; i++)
      printf("%s: ID number is %d.\n", cse406[i].name, cse406[i].id);
}
```

```
/* functions */

student getdata( int i )
{
   student temp;
   char inbuf[80];
   int j = 0;

   printf("Student #%d:\n", i);

   /* get the student name - then remove the newline character in the string */

   printf("      Name: ");
   fgets(temp.name, sizeof(temp.name), stdin);  /* note how data is obtained from stdin */
   while ( *(temp.name + j) != '\n')
      j++;
   *(temp.name + j) = '\0';

   /* get the student ID number */

   printf("      ID #: ");
   fgets(inbuf, sizeof(inbuf), stdin);
   sscanf(inbuf, "%d", &temp.id);

   /* return the structure */

   return temp;
}

thor> gcc -o structures structures.c
thor> structures
Number of students: 4
Student #0:
      Name: Allen Apple
      ID #: 12345
Student #1:
      Name: Irving Illini-Sweetcorn
      ID #: 78924
Student #2:
      Name: Frieda Feta-Cheese
      ID #: 78627
Student #3:
      Name: Zach Zucchini
      ID #: 96278
Allen Apple: ID number is 12345.
Irving Illini-Sweetcorn: ID number is 78924.
Frieda Feta-Cheese: ID number is 78627.
Zach Zucchini: ID number is 96278.
thor> exit
```

```c
/*
 *  dijkstra.c  -  John K. Estell  -  19 June 1995
 *  Implementation of Dijkstra's Algorithm.
 *  Vertices are v[0] through v[n-1], with n being the number of verice present in the
 *  graph.  Will find shortest path from v[0] to v[z].  n will be obtained dynamically.
 *  For sets, index refers to vertex, value of zero implies not a member of the set,
 *  non-zero implies membership.  For edges, enter weight if connected, "Inf" if not
 *  connected.
 */

#include <stdio.h>

/* function prototypes */

int get_integer(char *prompt);
float get_float(char *prompt);
void *make_array(int number_of_elements, int data_size);
int get_number_of_vertices(void);
int get_terminal_vertex(int number_of_vertices);

/* main */

int main(void)
{
   float *length,                    /* length of path from v[0] to v[i] */
         **weight;                   /* weight of edge {v[i], v[j]}      */
   float min_value,                  /* used for finding minimal length  */
         Inf = 1.0 / 0.0;            /* need to define infinity          */
   int   *set;                       /* distinguished set of vertices    */
   int   *path;                      /* used to mark minimal length path */
   int   *stack;                     /* used to backtrack the path       */
   int   n,                          /* number of vertices               */
         u,                          /* minimal vertex                   */
         z,                          /* terminal vertex                  */
         i, j;                       /* generic counting variables       */
   char  outbuf[132];

   /* get input */

   n = get_number_of_vertices();
   z = get_terminal_vertex(n);

   /* allocate memory for one-dimensional arrays */

   length = (float *) make_array(n, sizeof(float));
   set = (int *) make_array(n, sizeof(int));
   path = (int *) make_array(n, sizeof(int));
   stack = (int *) make_array(n, sizeof(int));

   /* allocate memory for two-dimensional array */

   weight = (float **) make_array(n, sizeof(float *));
   for (i = 0; i < n; i++)
      weight[i] = (float *) make_array(n, sizeof(float));
```

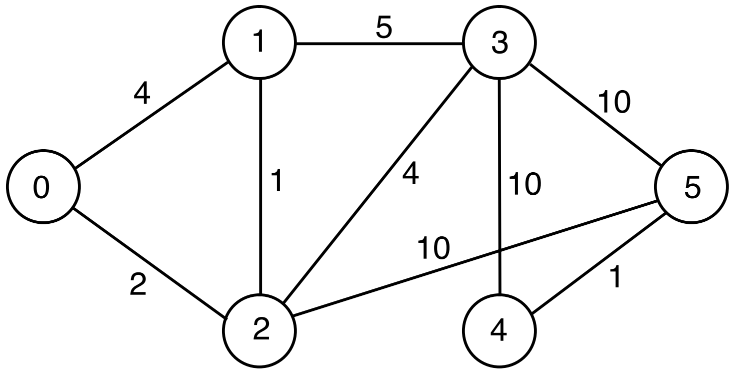
```
/* get the weights */

for (i = 0; i < n; i++)
   for (j = 0; j < n; j++)
   {
      sprintf(outbuf, "Enter weight w(%d,%d): ", i, j);
      weight[i][j] = get_float(outbuf);
   }

/*
 *  initialize Dijkstra's Algorithm variables:
 *     the distinguished set of vertices is initially empty.
 *     length[0] = 0, all other lengths are set to infinity.
 */

for (i = 0; i < n; i++)
{
   set[i] = 0;
   path[i] = -1;
}

length[0] = 0.0;
for (i = 1; i < n; i++)
   length[i] = Inf;

/* Perform the iteration */

while ( set[z] == 0 )
{
   /* find minimal vertex u not in the set */

   min_value = Inf;
   for (i = 0; i < n; i++)
      if (set[i] == 0)
         if (length[i] < min_value)
         {
            min_value = length[i];
            u = i;
         }

   /* add the minimal vertex to the set (and print out info for demo) */

   printf("min vertex: %d\n", u);
   printf("min length: %f\n", length[u]);

   set[u] = 1;
```

```
      /* update the length values and the path back pointer */

      for (i = 0; i < n; i++)
         if (set[i] == 0)
         {
            if (length[u] + weight[u][i] < length[i])
            {
               length[i] = length[u] + weight[u][i];
               path[i] = u;
            }
         }
   }

   /* print out shortest path and its length */

   i = 0;
   while (u != 0)
   {
      stack[i++] = u;
      u = path[u];
   }
   stack[i] = 0;
   printf("Shortest path is: ");
   while (i >= 0)
      printf("%d ", stack[i--]);
   printf("\n");

   printf("Length of shortest path is %f\n", length[z]);
   return 0; /* program complete - successful termination */
}

/*
 *  get_integer - will obtain an integer from the standard input.
 *                function will not return until an integer is obtained.
 */

int get_integer(char *prompt)
{
   int value, valid_input = 0;
   char inbuf[133];

   while (valid_input < 1)
   {
      printf(prompt);
      fgets(inbuf, sizeof(inbuf), stdin);
      valid_input = sscanf(inbuf, "%d", &value);
      if (valid_input == -1)
         printf("Please enter an integer value.\n");
      if (valid_input == 0)
         printf("That is not an integer - please enter an integer value.\n");
   }

   return value;
}
```

```
/*
 *  get_float - will obtain a floating point value from the standard input.
 *              function will not return until a floating point value
 *              is obtained.
 */

float get_float(char *prompt)
{
   int valid_input = 0;
   float value;
   char inbuf[133];

   while (valid_input < 1)
   {
      printf(prompt);
      fgets(inbuf, sizeof(inbuf), stdin);
      valid_input = sscanf(inbuf, "%f", &value);
      if (valid_input == -1)
         printf("Please enter a floating point value.\n");
      if (valid_input == 0)
      {
         printf("That is not a floating point value.\n");
         printf("Please enter a floating point value.\n");
      }
   }

   return value;
}

/*
 *  make_array - will allocate memory for specified number of cells of
 *               specified size.  Returns address of allocation, or
 *               forces program termination on insufficient memory.
 */

void *make_array(int number_of_elements, int data_size)
{
   void *address;

   address = (void *) malloc(number_of_elements * data_size);
   if (address == NULL)
   {
      fprintf(stderr, "Insufficient memory for array allocation.\n");
      exit(1);
   }

   return address;
}
```

```
/*
 *  get_number_of_vertices - get the number of vertices to be used.
 */

int get_number_of_vertices(void)
{
   int number = -1;

   while (number < 2)
   {
      number = get_integer("Enter number of vertices: ");
      if (number < 2)
         printf("The number of vertices must be 2 or greater.\n");
   }

   return number;
}

/*
 *  get_terminal_vertex - get vertex number of the terminal vertex.
 */

int get_terminal_vertex(int number_of_vertices)
{
   int number = -1;

   while ((number < 1) || (number > number_of_vertices - 1))
   {
      number = get_integer("Enter terminal vertex: ");
      if ((number < 1) || (number > number_of_vertices - 1))
         printf("The value of the terminal vertex must be between 1 and %d.\n",
                 number_of_vertices - 1);
   }

   return number;
}
```

```
thor> dijkstra
Enter number of vertices: 1000000000
Enter terminal vertex: 3
Insufficient memory for array allocation.
thor> dijkstra
Enter number of vertices: three
That is not an integer - please enter an integer value.
Enter number of vertices:
Please enter an integer value.
Enter number of vertices: 3
Enter terminal vertex: -1
The value of the terminal vertex must be between 1 and 2.
Enter terminal vertex: 4
The value of the terminal vertex must be between 1 and 2.
Enter terminal vertex: ^C
thor>
thor> dijkstra
Enter number of vertices: 6
Enter terminal vertex: 4
Enter weight w(0,0): Inf
Enter weight w(0,1): 4
Enter weight w(0,2): 2
Enter weight w(0,3): Inf
Enter weight w(0,4): Inf
Enter weight w(0,5): Inf
Enter weight w(1,0): 4
Enter weight w(1,1): Inf
Enter weight w(1,2): 1
Enter weight w(1,3): 5
Enter weight w(1,4): Inf
Enter weight w(1,5): Inf
Enter weight w(2,0): 2
Enter weight w(2,1): 1
Enter weight w(2,2): Inf
Enter weight w(2,3): 4
Enter weight w(2,4): Inf
Enter weight w(2,5): 10
Enter weight w(3,0): Inf
Enter weight w(3,1): 5
Enter weight w(3,2): 4
Enter weight w(3,3): Inf
Enter weight w(3,4): 10
Enter weight w(3,5): 10
Enter weight w(4,0): Inf
Enter weight w(4,1): Inf
Enter weight w(4,2): Inf
Enter weight w(4,3): 10
Enter weight w(4,4): Inf
Enter weight w(4,5): 1
Enter weight w(5,0): Inf
Enter weight w(5,1): Inf
Enter weight w(5,2): 10
Enter weight w(5,3): 10
Enter weight w(5,4): 1
Enter weight w(5,5): Inf
min vertex: 0
min length: 0.000000
min vertex: 2
min length: 2.000000
min vertex: 1
min length: 3.000000
min vertex: 3
min length: 6.000000
min vertex: 5
min length: 12.000000
min vertex: 4
min length: 13.000000
Shortest path is: 0 2 5 4
Length of shortest path is 13.000000
thor>
```



```
/*
 *  intarray.h  -  John K. Estell  -  15 June 1995
 *  generic functions for handling unsigned long integer arrays.
 */

typedef unsigned long int u_long;


/* function prototypes */

u_long max_value(u_long *array, int array_size);
u_long array_sum(u_long *array, int array_size);
int    max_value_index(u_long *array, int array_size);
void   init_array(u_long *array, int array_size, u_long value);


/* functions */


/*
 *  max_value - return contents of integer array element having
 *              the maximum value in the set.
 */

u_long max_value(u_long *array, int array_size)
{
   u_long maximum = 0;
   int i;

   for (i = 0; i < array_size; i++)
      if (maximum < array[i])
         maximum = array[i];

   return maximum;
}


/*
 *  array_sum - returns the sum of the contents of an array.
 */

u_long array_sum(u_long *array, int array_size)
{
   u_long sum = 0;
   int i;

   for (i = 0; i < array_size; i++)
      sum += array[i];

   return sum;
}
```

```
/*
 *  max_value_index - returns the index of the element of the array
 *                    having the maximum value in the set.
 */

int max_value_index(u_long *array, int array_size)
{
   u_long maximum = 0;
   int index = 0;
   int i;

   for (i = 0; i < array_size; i++)
      if (maximum < array[i])
      {
         maximum = array[i];
         index = i;
      }

   return index;
}


/*
 *  init_array - initialize the contents of an array to a specified value.
 */

void init_array(u_long *array, int array_size, u_long value)
{
   int i;

   for (i = 0; i < array_size; i++)
      array[i] = value;
}
```

conditional frequency - uses integer array package

```
/*
 *  condfreq.c  -  John K. Estell  -  16 June 1995
 *  Determine the conditional frequency of letters in the
 *  English language.
 *  The "intarray.h" file included below provides generic
 *  functions for one-dimensional unsigned long integer arrays.
 *  The use of include files such as this allows us to easily
 *  reuse code from one program to the next...
 */

#include <stdio.h>
#include <ctype.h>
#include "intarray.h"

#define SPACE ' '
#define SIZE 27

/* function prototypes */

int    character_index(char ch);
char   character(int index);
void   report_fatal_error(char *item, char *reason);

/* main */

int main(int argc, char *argv[])
{
   u_long count[SIZE][SIZE];
   FILE *fp;
   int i, j;
   char current, previous = SPACE;

   /* open the file if there is one... */

   switch (argc)
   {
      case 1: fp = stdin;
              break;

      case 2: fp = fopen(argv[1], "r");
              if (fp == NULL)
                 report_fatal_error(argv[0], "can't open file");
              break;

      default: report_fatal_error(argv[0], "too many arguments");
   }

   /* initialize the array */

   for (i = 0; i < SIZE; i++)
      init_array(count[i], SIZE, 0);
   /* process the input stream */

   while ((current = fgetc(fp)) != EOF)
   {
      count[character_index(previous)][character_index(current)]++;
      previous = current;
   }

   /* remove the space-space condition from the evaluation */

   count[character_index(SPACE)][character_index(SPACE)] = 0;

   /* determine the results */

   for (i = 0; i < SIZE; i++)
      printf("'%c' - followed by '%c' %5.2f%% of the time.\n",
              character(i),
              character(max_value_index(count[i], SIZE)),
             100.0 * (float) max_value(count[i], SIZE) / (float) array_sum(count[i], SIZE)
            );
}

/* *** functions *** */

/*  character_index - return the array index for the specified character. */

int character_index(char ch)
{
   if (isalpha(ch))
      return (toupper(ch) - 'A');
   else
      return (SIZE - 1);
}


/* character - return the appropriate character for the specified index. */

char character(int index)
{
   if (index < SIZE - 1)
      return ('A' + index);
   else
      return SPACE;
}


/* report_fatal_error - reports the error, then terminates execution. */

void report_fatal_error(char *item, char *reason)
{
   fprintf(stderr, "%s: %s\n", item, reason);
   exit(1);
}
```

```
thor> condfreq badfile
condfreq: can't open file
thor> condfreq file1 file2
condfreq: too many arguments
thor> wc ~cse305/jargon.txt
   21549  171169 1125795 /home/jupiter/cse305/jargon.txt
thor> condfreq ~cse305/jargon.txt
'A' - followed by 'N' 15.98% of the time.
'B' - followed by 'E' 22.97% of the time.
'C' - followed by 'O' 20.50% of the time.
'D' - followed by ' ' 50.49% of the time.
'E' - followed by ' ' 32.81% of the time.
'F' - followed by ' ' 32.15% of the time.
'G' - followed by ' ' 35.23% of the time.
'H' - followed by 'E' 40.13% of the time.
'I' - followed by 'N' 27.14% of the time.
'J' - followed by 'O' 23.59% of the time.
'K' - followed by 'E' 36.95% of the time.
'L' - followed by 'E' 18.15% of the time.
'M' - followed by 'E' 20.83% of the time.
'N' - followed by ' ' 28.07% of the time.
'O' - followed by 'N' 17.56% of the time.
'P' - followed by 'E' 17.21% of the time.
'Q' - followed by 'U' 93.57% of the time.
'R' - followed by ' ' 21.12% of the time.
'S' - followed by ' ' 39.69% of the time.
'T' - followed by 'H' 25.50% of the time.
'U' - followed by 'S' 18.86% of the time.
'V' - followed by 'E' 58.08% of the time.
'W' - followed by 'A' 20.09% of the time.
'X' - followed by ' ' 37.31% of the time.
'Y' - followed by ' ' 71.46% of the time.
'Z' - followed by 'E' 40.07% of the time.
' ' - followed by 'T' 14.22% of the time.
thor>
```

```
/*
 *  advent.c  -  John K. Estell  -  16 June 1995
 *  Demonstration of playing around with structures
 *  by developing an adventure game.
 *
 *  This particular implementation is an example of an engine, where only
 *  the logic is contained; the actual implementation of the adventure
 *  game (with respect to the text and room connections) are read in from
 *  a text file passed as a command line argument.  The format of the file is:
 *     line 1: integer giving the number of rooms to be read in.
 *  Remaining lines describe the rooms:
 *     1st line: comment line used to label room:  *** ROOM 0 ***
 *     2nd line: integer pointing to room forward of current room.
 *     3rd line: integer pointing to room left of current room.
 *     4th line: integer pointing to room right of current room.
 *        - for 2nd through 4th lines: use -1 to indicate no connection
 *     5th line: start of text - maximum is 1000 characters.
 *        - starting from 5th line just type in the description text as you
 *          want it to appear in the output; newlines will automatically
 *          be handled.  Use "END" (without the quotes) at the beginning
 *          of a line to indicate the end of the description text.
 */

#include <stdio.h>
#include <string.h>

/*
 *  type declaration
 */

struct room_struct
{
   char *text;                   /* pointer to room description text */
   struct room_struct *forward;  /* pointer to next room forwards    */
   struct room_struct *left;     /* pointer to room to the left      */
   struct room_struct *right;    /* pointer to room to the right     */
};

typedef struct room_struct room;  /* the structure is now the type "room" */

/*
 *  function prototypes
 */

char get_user_input(void);
room get_room_information(FILE *fp, room *room_array);
room *move(room *current, room *direction);
void report_fatal_error(char *item, char *reason);
```

```
/*
 *  main
 */

main(int argc, char *argv[])
{
   room *current,    /* pointer to the current room           */
        *rooms;      /* array for all of the rooms to be used */
   FILE *fp;
   char buf[132];
   int  num_rooms,
        valid_read,
        i;

   /* check for correct number of arguments */

   if (argc != 2)
      report_fatal_error("Usage", strcat(argv[0], " adventurefile"));

   /* open specified adventurefile */

   fp = fopen(argv[1], "r");
   if (fp == NULL)
      report_fatal_error(argv[1], "error reading file");

   /* get number of rooms */

   fgets(buf, sizeof(buf), fp);
   valid_read = sscanf(buf, "%d", &num_rooms);
   if (valid_read < 1)
      report_fatal_error(argv[1], "error in file format");
   if (num_rooms < 1)
      report_fatal_error(argv[1], "invalid room number value");

   /* allocate space for rooms */

   rooms = (room *) malloc( num_rooms * sizeof(room) );
   if (rooms == NULL)
      report_fatal_error(argv[1], "insufficient memory for number of rooms");

   /* get room information */

   for (i = 0; i < num_rooms; i++)
      rooms[i] = get_room_information(fp, rooms);

   /* finished with adventurefile so close it */

   fclose(fp);
   /* point to starting room and welcome the player */

   current = &rooms[0];
   printf("Welcome to advent!\n\n");
   printf("Use 'f' to go forward, 'l' to go left,\n");
   printf("'r' to go right, and 'q' to quit.\n\n");

   /* display initial room description */

   printf(current -> text);  /* print out text description for the current room */

   /* let the player run through the rooms... */

   while (1)
   {
      switch ( get_user_input() )
      {
         case 'F':
         case 'f': current = move(current, current -> forward);
                   break;
         case 'L':
         case 'l': current = move(current, current -> left);
                   break;
         case 'R':
         case 'r': current = move(current, current -> right);
                   break;
         case 'Q':
         case 'q': printf("Thanks for playing!\n");
                   exit(0); /* program termination */
                   /* NOT REACHED */

         default: printf("I don't understand that.\n");
      }
   }
}
```

```
/*
 *  functions
 */


/*
 *  report_fatal_error: prints out an error message and terminates program.
 */

void report_fatal_error(char *item, char *reason)
{
   fprintf(stderr, "%s: %s\n", item, reason);
   exit(1);
}


/*
 *  get_user_input: prints prompt, returns character input from user
 *                  to indicate command to execute.
 */

char get_user_input(void)
{
   char ch;
   char inbuf[132];

   printf("Command> ");
   fgets(inbuf, sizeof(inbuf), stdin);
   sscanf(inbuf, " %c", &ch);
   return ch;
}


/*
 *  move: used to determine is a movement is possible (i.e. a link to the
 *        indicated direction exists) and if so, to move there and print
 *        out the new room's descriptive text.  If to link in that
 *        direction exists, print out an appropriate message.  Function
 *        will return a pointer to the current room.
 */

room *move(room *current, room *direction)
{
   if (direction != NULL)
   {
      printf(direction -> text);
      return direction; /* points to the new room */
   }
   else
   {
     printf("You can't go in that direction.\n");
     return current;  /* points to the current room */
   }
}
```

```
/*
 *  get_room_information: used to parse one record of room information
 *                        from the specified input file.  Format of the
 *                        input file is given in the header documentation.
 */

room get_room_information(FILE *fp, room *room_array)
{
   char inbuf[133], stringbuf[1000];
   int  room_ptr;
   room temp;

   fgets(inbuf, sizeof(inbuf), fp);  /* read and discard room header info */

   /* set up the pointers to the other rooms... */

   fgets(inbuf, sizeof(inbuf), fp);  /* get forward room */
   sscanf(inbuf, "%d", &room_ptr);
   if (room_ptr >= 0)
        temp.forward = &room_array[room_ptr];
   else
      temp.forward = NULL;

   fgets(inbuf, sizeof(inbuf), fp);  /* get left room */
   sscanf(inbuf, "%d", &room_ptr);
   if (room_ptr >= 0)
        temp.left = &room_array[room_ptr];
   else
      temp.left = NULL;

   fgets(inbuf, sizeof(inbuf), fp);  /* get right room */
   sscanf(inbuf, "%d", &room_ptr);
   if (room_ptr >= 0)
        temp.right = &room_array[room_ptr];
   else
      temp.right = NULL;

   /* handle the descriptive room text... */

   stringbuf[0] = '\0';   /* initialize text string */
   while (fgets(inbuf, sizeof(inbuf), fp), strncmp(inbuf, "END", 3)) /* note comma expr. */
      strcat(stringbuf, inbuf);
   temp.text = (char *) malloc( 1 + strlen(stringbuf) );  /* add 1 for NUL character */
   strcpy(temp.text, stringbuf);  /* store the results */

   /* return the record */

   return temp; /* pointer to the new record allocated in memory */
}
```

```
thor> cat advent.dat
4
*** ROOM 0 ***
1
3
-1
You are in the cafeteria, where several students are seen crawling to the
stomach pump after having eaten the Chef’s Surprise.  Peering through the
door to the kitchen, you notice several clumps of cat hair on the butchers
block.  In the background, you hear the chef saying,
  "Prepared properly, cat tastes just like chicken - and it’s cheaper!"
END
*** ROOM 1 ***
-1
3
2
There are three witches before you, all slowly chanting in unison:
  Mortal, we have summon thee, make haste!
  And go forth into the farrow’d waste.
  Find eye of newt, and toe of frog,
  And deliver thus to this Scottish bog.
  These things we need t’ brew our charm;
  Bring them forth - and suffer no ’arm.
  Leave us and go!
  ’Tis no more to be said,
  Save if you fail, then thou be stricken, dead.
END
*** ROOM 2 ***
0
-1
-1
You’re at the entrance to Pokagon State Park.  The employee at the
gate is staring at you - I think she wants something.
END
*** ROOM 3 ***
2
0
1
You’re in class working on the Collegian crossword puzzle.
You’ve missed Dr. Estell’s historical commentary on the adventure
game assignments he has given in his assembly language classes - lucky you!
END
```

```
thor> advent
Usage: advent adventurefile
thor> advent badfile
badfile: error reading file
thor> advent advent.c
advent.c: error in file format
thor> advent advent.dat
Welcome to advent!

Use 'f' to go forward, 'l' to go left,
'r' to go right, and 'q' to quit.

You are in the cafeteria, where several students are seen crawling to the
stomach pump after having eaten the Chef's Surprise.  Peering through the
door to the kitchen, you notice several clumps of cat hair on the butchers
block.  In the background, you hear the chef saying,
  "Prepared properly, cat tastes just like chicken - and it's cheaper!"
Command> s
I don't understand that.
Command> l
You're in class working on the Collegian crossword puzzle.
You've missed Dr. Estell's historical commentary on the adventure
game assignments he has given in his assembly language classes - lucky you!
Command> l
You are in the cafeteria, where several students are seen crawling to the
stomach pump after having eaten the Chef's Surprise.  Peering through the
door to the kitchen, you notice several clumps of cat hair on the butchers
block.  In the background, you hear the chef saying,
  "Prepared properly, cat tastes just like chicken - and it's cheaper!"
Command> l
You're in class working on the Collegian crossword puzzle.
You've missed Dr. Estell's historical commentary on the adventure
game assignments he has given in his assembly language classes - lucky you!
Command> r
There are three witches before you, all slowly chanting in unison:
  Mortal, we have summon thee, make haste!
  And go forth into the farrow'd waste.
  Find eye of newt, and toe of frog,
  And deliver thus to this Scottish bog.
  These things we need t' brew our charm;
  Bring them forth - and suffer no 'arm.
  Leave us and go!
  'Tis no more to be said,
  Save if you fail, then thou be stricken, dead.
Command> f
You can't go in that direction.
Command> r
You're at the entrance to Pokagon State Park.  The employee at the
gate is staring at you - I think she wants something.
Command> Q
Thanks for playing!
thor> exit
```

```
/*
 *  lower.c  -  John K. Estell  -  12 February 1992
 *  use low-level i/o to do lowercase conversion routine
 *  format is: lower infile outfile
 */

#include <stdio.h>
#include <sys/file.h>
#include <errno.h>
#include <ctype.h>

#define BUFSIZ 1024

main(int argc, char *argv[])
{
  char buffer[BUFSIZ];
  int i, bytes_read, bytes_written;
  int fd1, fd2;

  /* open input file, create output file... */

  if ((fd1 = open(argv[1], O_RDONLY)) == -1) /* then a read error occurred */
  {
    fprintf(stderr, "%s: cannot open %s\n", argv[0], argv[1]);
    perror("");  /* prints system error message */
    exit(1);
  }
  if ((fd2 = open(argv[2], (O_CREAT | O_EXCL | O_WRONLY), 0640)) == -1) /* have error... */
  {
    fprintf(stderr, "%s: cannot open %s\n", argv[0], argv[2]);
    perror("");  /* prints system error message */
    exit(1);
  }

  /* process the file */

  while ((bytes_read = read(fd1, buffer, BUFSIZ)) > 0)  /* read from file */
  {
    for (i = 0; i < bytes_read; i++)
      buffer[i] = tolower(buffer[i]);
    bytes_written = write(fd2, buffer, bytes_read);          /* write to file */
    if (bytes_written == -1)
      break;
  }
  if ((bytes_read == -1) || (bytes_written == -1)) /* had a fatal error, so bail out... */
  {
    perror(argv[0]);
    exit(1);
  }

  close(fd1);  /* close the two files */
  close(fd2);
  exit(0);
}
```

```
Script started on Wed Feb 12 08:39:11 1992
```
{newlines and comments added for clarity}

{below is the text to be processed}

```
Thor> cat text
  This is some sample text to illustrate the successful operation
of the lower.c program that demonstrates the use of LOW-LEVEL I/O.

  THIS SENTENCE IS WRITTEN IN THE ORIGINAL TEXT FILE COMPLETELY IN
UPPERCASE LETTERS!

  Upon successful operation of the lower.c program, the above line
will be in lower case, as will the first letter in the first word
of this sentence, which seems to be running on and on and on for
NO PARTICULAR REASON!!!

  Well, I reckon that this presents enough of a demonstration to
convince you all that this works.  Want to go watch the PIG RACES?
That's a major source of entertainment in Central Illinois.....
```

{process the text}

```
Thor> lower text text.out
```

{look at the results}

```
Thor> cat text.out
  this is some sample text to illustrate the successful operation
of the lower.c program that demonstrates the use of low-level i/o.

  this sentence is written in the original text file completely in
uppercase letters!

  upon successful operation of the lower.c program, the above line
will be in lower case, as will the first letter in the first word
of this sentence, which seems to be running on and on and on for
no particular reason!!!

  well, i reckon that this presents enough of a demonstration to
convince you all that this works.  want to go watch the pig races?
that's a major source of entertainment in central illinois.....
```

{examples of error messages from use of improper filenames}

```
Thor> lower badfile text.out2
lower: cannot open badfile
No such file or directory

Thor> lower text text.out
lower: cannot open text.out
File exists

Thor> exit
```

```
/*
 * screensize.c  -  John K. Estell  -  1 February 1993
 *
 * Demonstration of the curses package for obtaining
 * the screen size of the current display.
 *
 * program must be compiled as follows:
 *    gcc -o screensize screensize.c -lcurses -ltermcap
 * in order to link in the appropriate libraries.
 */

#include <stdio.h>
#include <signal.h>
#include <curses.h>

main()
{
  /* create a window - initialize screen and get terminal characteristics */

  initscr();

  /* give line and column information for current display */

  printf("Number of lines on this terminal are %d.\n", LINES);
  printf("Number of columns on this terminal are %d.\n", COLS);
}
```

```
/*
 *  emacs1.c  -  John K. Estell  -  2 February 1993
 *  example of curses use:  create window, select raw mode, move about
 *  using control characters, and exit on ctrl-c
 *  to compile: gcc emacs1.c -o emacs1 -lcurses -ltermcap
 */

#include <stdio.h>
#include <signal.h>
#include <curses.h>

main()
{
  int row = 0, col = 0;
  char c;

  initscr();    /* create a window */
  raw();        /* no characters, including CTRL-C, have special meaning */
  noecho();     /* do not echo characters as soon as they're typed */

  while (1)  /* deliberate infinite loop - make sure a break statement is inside! */
  {
    /* limit cursor to the current screen */

    if (row < 0)
      row = 0;
    if (row > LINES - 1)
      row = LINES - 1;            /* LINES defined and assigned value by curses package. */
    if (col < 0)
      col = 0;
    if (col > COLS - 1)
      col = COLS - 1;             /* COLS defined and assigned value by curses package. */

    /* move cursor to x-y coordinate and display changes on terminal */

    move(row, col);    /* move the cursor */
    refresh();         /* update the screen display */

    /* get input character - strip off MSB to be safe... */

    c = getch() & 0x7F;      /* usually this is not needed.... */
    /* process the character....  \0dd is 3-digit octal character code */

    if (c == '\006')         /* CTRL-F: move right (forward) */
      col++;
    else if (c == '\002')    /* CTRL-B: move left (backward) */
      col--;
    else if (c == '\016')    /* CTRL-N: move to next line (down) */
      row++;
    else if (c == '\020')    /* CTRL-P: move to previous line (up) */
      row--;
    else if (c == '\015')    /* CTRL-M: carriage return and line feed */
    {
      row++;
      col = 0;
    }
    else if (c == '\003')    /* CTRL-C: exit */
    {
      move(LINES-1, 0);      /* move cursor to bottom of screen */
      refresh();
      endwin();              /* end the window */
      printf("\n");          /* cleanup */
      exit();
    }
    else /* have a character to display on the screen.... */
    {
      insch(c);              /* insert the character on the screen */
      col++;                 /* and go to the next character location... */
    }
  }
}
```

pager - uses curses to display a file one screen page at a time

```
/*
 *  pager.c  -  John K. Estell  -  2 December 1995
 *  pager version 3.0 - page through the file with curses.  pager is a program that will
 *  allow a person to "page" through a text file.  In essence, it's a simplified version
 *  of the 'less' program.  Note: curses.h defines both TRUE and FALSE values used here.
 *  compile with: gcc -o pager pager.c -lcurses -ltermcap
 */


#include <stdio.h>
#include <curses.h>

struct page
   {
      struct page *previous;
      struct page *next;
      long int position;
   };

typedef struct page PAGE; /* gcc has problems if last two statements are combined... */

PAGE *current, *previous, *next, first;

char get_user_input(void);
void set_link(PAGE *link, long int position, PAGE *prev, PAGE *next);

main( int argc, char **argv )
{
   int num_rows = 0, num_cols = 0, num_pages = 1, page_count = 1;
   long int char_count = 0, file_length, end_of_page;
   int ch, i, good_input, row_count;
   FILE *fp;

   /* open input file */

   if (argc != 2) /* then wrong number of parameters were passed */
   {
      fprintf(stderr, "Usage: %s filename\n", argv[0]);
      exit(1);
   }
   if ((fp=fopen(argv[1], "r")) == NULL)
   {
      fprintf(stderr, "%s: file not found.\n", argv[1]);
      exit(1);
   }

   initscr();   /* start curses package - get terminal information */

   /* initialize linked list */

   current = &first;  /* point to the first page */
   set_link(&first, 0, NULL, NULL);
```

```
/* process file */

while ((ch = getc(fp)) != EOF)
{
   char_count++;
   num_cols++;

   /*
    *  Check for newline condition.  This can happen either from
    *  a newline character or by reaching the last column in the display
    */

   if ((ch == '\n') || (num_cols == COLS))
   {
      num_rows++;
      num_cols = 0;

      /*
       *  On newline check for newpage condition.  When a new page is
       *  needed, create a new pagemarker record and set up the links.
       *  NOTE: this does not check for the boundary condition of a
       *  file ending on the last character of a page!
       */

      if ((num_rows % (LINES - 1)) == 0)
      {
         num_pages++;
         next = (PAGE *) malloc(sizeof(PAGE));
         if (next == NULL)
         {
            perror(argv[0]);
            exit(1);
         }
         current->next = next;
         previous = current;
         current = next;
         set_link(current, char_count, previous, NULL);
      }
   }
}
```

```
/*
 *  Test routine: display the information in the file
 *  space bar - next page
 *  minus sign - previous page
 *  q - quit
 */

/* initialize curses package */

raw();         /* do not buffer input characters */
noecho();      /* do not echo input */

/* point to start of file... */

current = &first;
file_length = char_count++;

while (1)
{
   if (current->next == NULL)
      end_of_page = file_length;
   else
      end_of_page = (current->next)->position;
   fseek(fp, current->position, 0); /* move to current position in file */
   clear();

   for (i = current->position; i < end_of_page; i++)
   {
      ch = getc(fp);
      addch(ch);     /* write the character to the screen */
   }

   if (current->next == NULL)
      move(LINES - 1,0);         /* move to bottom of last page */

   standout();
   printw(" -- %s: page %d of %d -- ", argv[1], page_count, num_pages);
   standend();
   refresh();  /* update screen display */

   ch = get_user_input();

   if (ch == 'q')
   {
      clear();
      refresh();
      endwin();    /* end curses window and restore terminal settings */
      fclose(fp);  /* close the file */
      exit(0);
   }
```

```
      if (ch == ' ')
      {
         current = current->next;
         page_count++;
      }
      if (ch == '-')
      {
         current = current->previous;
         page_count--;
      }
   }
}


char get_user_input(void)
{
   int good_input = FALSE;
   char ch;

   while (!good_input)
   {
      ch = tolower(getch());
      if (ch == 'q')
         good_input = TRUE;
      if ((ch == ' ') && (current->next != NULL))
         good_input = TRUE;
      if ((ch == '-') && (current->previous != NULL))
         good_input = TRUE;

      if (!good_input)
      {
         putc('\a', stdout);  /* ring the bell to annoy the user */
         fflush(stdout);      /* must flush the output buffer in order to do this */
      }
   }

   return ch;
}

void set_link(PAGE *link, long int position, PAGE *prev, PAGE *next)
{
   link->position = position;
   link->previous = prev;
   link->next     = next;
}
```

mymore - a different type of screen page-at-a-time display

```
/*
 *  mymore.c
 *  more demonstration program  -  John K. Estell  -  8 February 1993
 *  Documentation - Jim Kiraly - 2/8/93
 *
 *  This program uses curses and the terminal device file for
 *  handling terminal input while (possibly) redirecting standard input.
 */


#include <stdio.h>
#include <curses.h>

#define MORE    "-- more --"
#define NO_MORE "          "

main(int argc, char *argv[])
{
  int  ch;
  int  numlines = 1, LINES = 20;
  FILE *fp = stdin;
  FILE *fpin;
  FILE *fpout;

  if (argc == 2)
    fp = fopen(argv[1], "r");           /*  assume argv[1] is input file   */

  fpin  = fopen("/dev/tty", "r");       /*  Open the terminal device file  */
  fpout = fopen("/dev/tty", "w");       /*  for both reading and writing.  */

  savetty();                    /*  Save the current terminal settings.    */
  cbreak();                     /*  Character at a time input.             */
  noecho();                     /*  Characters are not to be echo back.    */

  while (( ch = fgetc(fp) ) != EOF)
  {
    fprintf(fpout, "%c", ch);   /*  Print the characters to the terminal   */
                                /*  device file.                           */
    if (ch == '\n')
      numlines++;
    if (numlines == LINES)
    {
      fprintf(fpout, MORE);
      fflush(fpout);            /*  Force the prompt out to the screen     */
                                /*  without using a carraige return.       */

      ch = fgetc(fpin);         /*  Read a character from the terminal     */
                                /*  device file.                           */
      fprintf(fpout, "\r");     /*  Erase the more prompt.                 */
      fprintf(fpout, NO_MORE);
      fprintf(fpout, "\r");
      fflush(fpout);
      numlines = 1;
      if ((ch == 'Q') || (ch == 'q'))
      {
        fclose(fp);
        fclose(fpin);
        fclose(fpout);
        echo();         /*  Turn on echo and turn off cbreak manually.    */
        nocbreak();
        resetty();      /*  Reset the terminal to the previous setting.   */
        exit();
      }                 /*  The calls to echo() and nocbreak() should not */
    }                   /*  be needed.  resetty() should take care of     */
  }                     /*  turning echo on and cbreak off.  The calls    */
                        /*  are a conservative way to make sure that      */
                        /*  everything is taken care of properly.         */
  fclose(fp);
  fclose(fpin);
  fclose(fpout);
  echo();
  nocbreak();
  resetty();
  exit();
}
```

- demonstration of use:

```
thor> mymore mymore.c
/*
 *  mymore.c
 *  more demonstration program  -  John K. Estell  -  8 February 1993
 *  Documentation - Jim Kiraly - 2/8/93
 *
 *  This program uses curses and the terminal device file for
 *  handling terminal input while redirecting standard input.
 */


#include <stdio.h>
#include <curses.h>

#define MORE    "-- more --"
#define NO_MORE "          "

main(int argc, char *argv[])
{
  int  ch;
-- more --            int  numlines = 1, LINES = 20;
  FILE *fp;
```

(remainder of output omitted)

- Note that the output of the program as displayed on the screen will overwrite the more prompt before continuing on. The script facility will essentially ignore the carriage return ('\r') character when recording the session.

emacs3 - primitive screen editor example

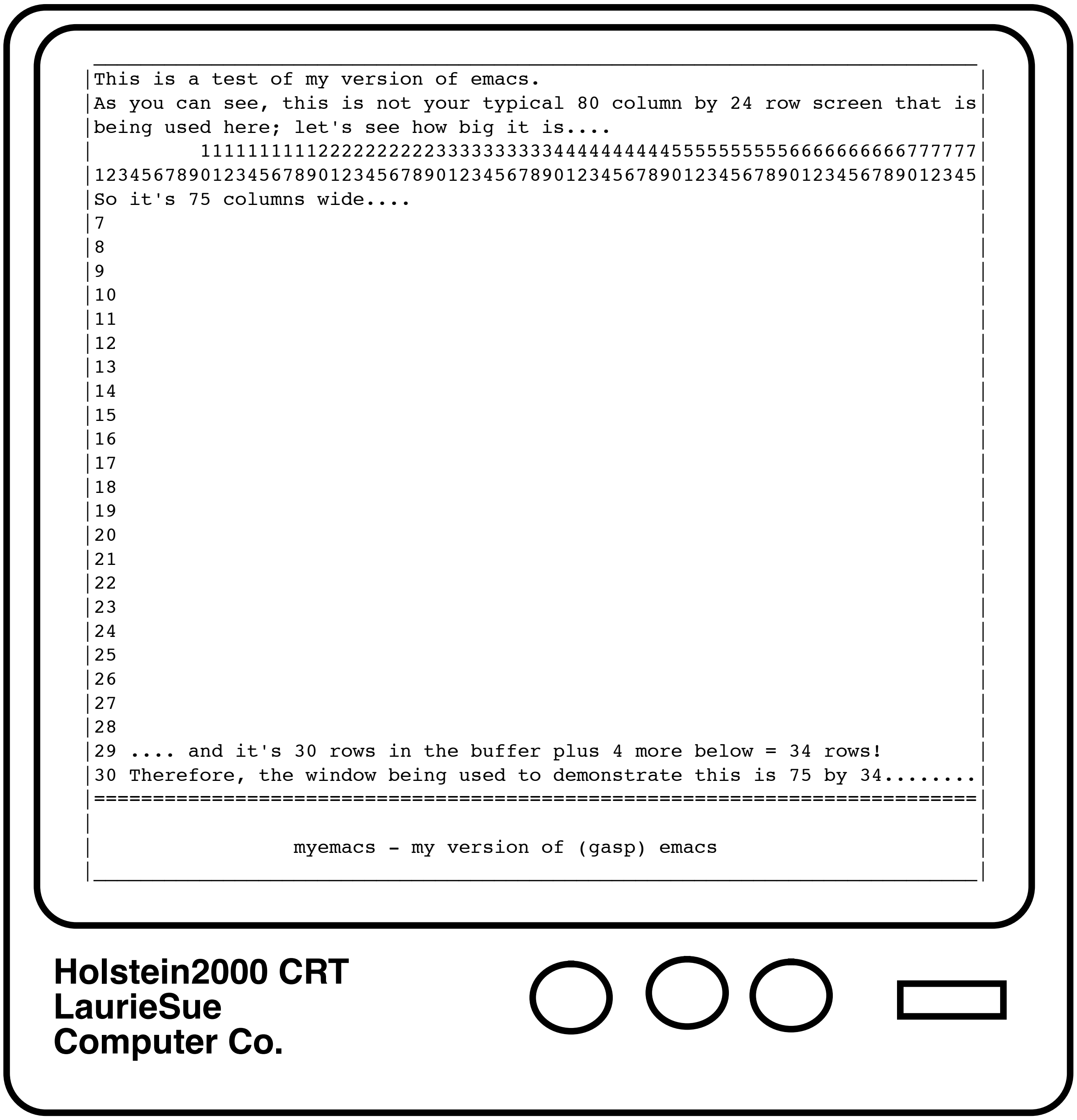
```
/*
 * emacs3.c - 10 June 1993
 *
 * This program dynamically allocates a buffer for storing data to demonstrate
 * how to manipulate what is supposed to be a two dimensional buffer in this
 * sort of situation.  The functions 'inch()' and 'winch(WINDOW *win)' return
 * the character located at the current cursor coordinates in the specified
 * window; the use of one of these functions would be more appropriate.
 * FOR MORE INFORMATION PLEASE SEE THE SUN OS MANUAL:
 *      "Programmer's Overview Utilities & Libraries", pp. 265-288
 */

#include <stdio.h>
#include <signal.h>
#include <curses.h>

/* GLOBALS */

int row, col;       /* positioning variables */
int ROW, COL;       /* limits on screen display */
char *buffer;       /* character storage buffer */
WINDOW *mywin;      /* predefined data type - working window */

/* function prototypes */

void endpgm(void);
void storebuf(int yrow, int xcol, char ch);
char getbuf(int yrow, int xcol);
void deleteline(void);
void insertline(void);
void savefile(void);
void initialize(void);
```

```
/* main */

main()
{
  char c;
  initialize();   /* initialize everything with this function call */

  while (1)
  {
    if (row < 0) row = 0;     /* failsafe cursor movement */
    if (row > ROW) row = ROW;
    if (col < 0) col = 0;
    if (col > COL) col = COL;

    wmove(mywin, row, col); /* move cursor and update screen display */
    wrefresh(mywin);
    c = getch() & 0x7F;     /* safety feature - usually not needed */

    if (c == '\006')        /* CTRL-F: move forward (right) */
      col++;
    else if (c == '\002')   /* CTRL-B: move backward (left) */
      col--;
    else if (c == '\016')   /* CTRL-N: move to next line (down) */
      row++;
    else if (c == '\020')   /* CTRL-P: move to previous line (up) */
      row--;
    else if (c == '\015')   /* CTRL-M: carriage return - line feed */
    {
      row++;
      col = 0;
    }
    else if (c == '\001')   /* CTRL-A: carriage return */
      col = 0;
    else if (c == '\013')   /* CTRL-K: delete line */
      deleteline();
    else if (c == '\017')   /* CTRL-O: insert line */
      insertline();
    else if (c == '\023')   /* CTRL-S: save buffer into file */
      savefile();
    else if (c == '\003')   /* CTRL-C: end program */
      endpgm();
    else if (c > '\032')    /* Is c > CTRL-Z?  If so, print the character... */
    {
      waddch(mywin, c);
      storebuf(row, col, c);
      col++;
    }
  }
}
```

```
/* endprog: exit the program (handler routine AND function call) */

void endpgm( void )
{
  move(LINES-1, 0);
  refresh();
  endwin();          /* finish up window routines */
  printf("\n");
  exit(0);
}



/* storebuf: store passed character in the buffer at given location */

void storebuf(int yrow, int xcol, char ch)
{
  int location;
  location = yrow * COL + xcol;   /* calculate storage location, mapping 2D to 1D */
  buffer[location] = ch;
}



/* getbuf: get character stored in the buffer at the given location */

char getbuf(int yrow, int xcol)
{
  int location;
  location = yrow * COL + xcol;    /* calculate storage location, mapping 2D to 1D */
  return buffer[location];
}



/* deleteline */

void deleteline(void)
{
  int yrow, xcol;
  wdeleteln(mywin);                               /* delete line on screen */
  for (yrow = row; yrow < ROW - 1; yrow++)        /* delete line in buffer */
    for (xcol = col; xcol < COL; xcol++)
      storebuf(yrow, xcol, getbuf(yrow + 1, xcol));
  for (xcol = 0; xcol < COL; xcol++)              /* initialize last line in buffer */
    storebuf(ROW - 1, xcol, ' ');
  col = 0;
}
```

```
/* savefile: save contents of screen buffer to a file */

void savefile(void)
{
  int xcol, yrow;
  FILE *fd;
  char filename[50];

  /* prompt for name of file */

  move( ROW + 1, COL / 4);
  addstr( "File: ");
  refresh();

  /* get name of file */

  noraw();                   /* can't get string while in raw mode */
  echo();                    /* allow user to see what's being typed */
  getstr(filename);          /* CTRL-C will be trapped by signal while */
  raw();                     /* outside of raw mode */
  noecho();

  /* write to file */

  if ((fd = fopen(filename, "w")) != NULL)
  {
    for (yrow = 0; yrow < ROW; yrow++)
    {
      for (xcol = 0; xcol < COL; xcol++)
        putc(getbuf(yrow, xcol), fd);
      putc('\n', fd);
    }
    fclose(fd);

    move(ROW + 1, COL / 4);
    addstr("                                                  "); /* 50 spaces */
    refresh();
  }
  else
  {
    move(ROW + 1, COL / 4);
    addstr("File could not be written                         ");
    refresh();
  }
}
```

```
/* insertline */

void insertline(void)
{
  int yrow, xcol;
  winsertln(mywin);                                 /* insert line on screen */
  for (yrow = ROW - 2; yrow >= row; yrow--)         /* insert line in buffer */
    for (xcol = 0; xcol < COL; xcol++)
      storebuf(yrow + 1, xcol, getbuf(yrow, xcol));
  for (xcol = 0; xcol < COL; xcol++)                /* initialize line */
    storebuf(row, xcol, ' ');
  col = 0;
}


/* initialize screen and buffer... */

void initialize(void)
{
  int i;
  signal(SIGINT, endpgm);    /* CTRL-C ends program by redirecting the signal... */

  initscr();                 /* initialize windows */
  raw();                     /* take characters without pre-processing */
  noecho();                  /* don't echo characters as they're typed */

  ROW = LINES - 3;           /* define our edge of window parameters */
  COL = COLS - 1;

  buffer = (char *) malloc( ROW * COL );          /* allocate space for buffer */

  for (i = 0; i < ROW * COL; i++)                 /* set buffer to all blanks */
    buffer[i] = ' ';

  row = col = 0;                                  /* starting position of cursor */
  move(ROW, 0);
  for(i=0; i<COL; i++)
    insch('=');

  move(ROW+2, COLS/4);
  standout();                                     /* print in inverse mode */
  addstr("myemacs - my version of (gasp) emacs");
  standend();
  refresh();                                      /* display the window */

  mywin = subwin(stdscr, ROW, COL, 0, 0);         /* create a subwindow to work in */
}
```



```
 ___________________________________________________________________________
|This is a test of my version of emacs.                                     |
|As you can see, this is not your typical 80 column by 24 row screen that is|
|being used here; let's see how big it is....                               |
|         111111111122222222223333333333444444444455555555556666666666777777|
|123456789012345678901234567890123456789012345678901234567890123456789012345|
|So it's 75 columns wide....                                                |
|7                                                                          |
|8                                                                          |
|9                                                                          |
|10                                                                         |
|11                                                                         |
|12                                                                         |
|13                                                                         |
|14                                                                         |
|15                                                                         |
|16                                                                         |
|17                                                                         |
|18                                                                         |
|19                                                                         |
|20                                                                         |
|21                                                                         |
|22                                                                         |
|23                                                                         |
|24                                                                         |
|25                                                                         |
|26                                                                         |
|27                                                                         |
|28                                                                         |
|29 .... and it's 30 rows in the buffer plus 4 more below = 34 rows!        |
|30 Therefore, the window being used to demonstrate this is 75 by 34........|
|===========================================================================|
|                                                                           |
|                 myemacs - my version of (gasp) emacs                      |
|___________________________________________________________________________|
```

example of using signals

```
/*
 *  ticktock.c  -  John K. Estell  -  5 July 1996
 *  test signal handler routine
 */

#include <stdio.h>
#include <signal.h>

int clock;

void confirm(int signo);     /* signal handler routine */
extern char *sys_siglist[];  /* external list of signal names */

main()
{
  signal(SIGTSTP, confirm);  /* trap CTRL-Z */
  signal(SIGINT, confirm);   /* trap CTRL-C */

  for (clock = 20; clock > 0; clock--)
  {
    printf("%s...\n", clock % 2 ? "tock" : "tick");
    sleep(1);
  }
  printf("Liftoff!\n");
}


/* confirm - supply information to the user regarding exiting the program */

void confirm(int signo) /* kernel will pass the signal number it received... */
{
  printf("\n*** Signal received: %s\n", sys_siglist[signo]);
  printf("*** countdown aborted at T-minus %d seconds\n", clock);
  exit(0);
}

thor> gcc -o ticktock ticktock.c
thor> ticktock
tick...
tock...
^Z                                                    (CTRL-Z entered by user - character displayed by the kernel)
*** Signal received: Stopped
*** countdown aborted at T-minus 19 seconds
thor> ticktock
tick...
tock...
tick...
tock...
^C                                             (CTRL-C entered by user - character displayed by the kernel)
*** Signal received: Interrupt
*** countdown aborted at T-minus 17 seconds
thor>
```

```
/*
 *  fork.c  -  John K. Estell  -  10 February 1992
 *  Test program to illustrate the use of the fork() system call.
 *  Executing the fork will cause the program to replicate itself,
 *  forming a child process to the parent process that caused the
 *  fork.  The child process starts execution with the statement
 *  following the fork().
 */

#include <stdio.h>

main()
{
  int pid;
  printf("Hello\n");   /* used to show that the child doesn't execute this */
  pid = fork();        /* "called once - returns twice": child process is created here */

  if (pid == -1) /* then the fork was unsuccessful */
  {
    fprintf(stderr, "Bad fork - use a spoon.\n");
    exit(1);
  }

  if (pid != 0) /* have parent process */
    printf("This is the parent process\n");  /* parent process has pid != 0 */
  else /* have child process */
    printf("This is the child process\n");   /* child process has pid == 0 */
}
Thor> gcc fork.c
Thor> a.out
Hello
This is the child process
This is the parent process
Thor> a.out
Hello
This is the child process
This is the parent process
Thor> a.out
Hello
This is the parent process
Thor> This is the child process
a.out
Hello
This is the child process
This is the parent process
Thor> a.out
Hello
This is the child process
This is the parent process
Thor> a.out
Hello
This is the parent process
This is the child process
Thor>
```

Note this time parent ended before child... .
and the shell printed its prompt before the child printed its message.

fork with parent-child synchronization

```
/*
 *  fork1.c  -  John K. Estell  -  10 February 1992
 *  Test program to illustrate fork using synchronization
 */

#include <stdio.h>
#include <sys/wait.h>

main()
{
  int pid;
  int status;

  printf("Hello\n");
  pid = fork();
  if (pid == -1)
  {
    fprintf(stderr, "Bad fork - use a knife.\n");
    exit(1);
  }
  if (pid)
  {
    wait(&status);  /* wait for the child process to end. */
    printf("This is the parent process\n");
  }
  else
    printf("This is the child process\n");
}
```

```
thor> gcc fork1.c
thor> a.out
Hello
This is the child process
This is the parent process
thor> a.out
Hello
This is the child process
This is the parent process
thor> a.out
Hello
This is the child process
This is the parent process
thor> a.out
Hello
This is the child process
This is the parent process
thor> a.out
Hello
This is the child process
This is the parent process
thor>
```

```
/*
 *  fork2.c  -  John K. Estell  -  10 February 1992
 *  test program to illustrate use of system call function
 */

#include <stdio.h>
#include <sys/wait.h>  /* needed when using wait() */

main()
{
  int pid;
  int status;
  printf("Hello\n");
  pid = fork();
  if (pid == -1)
  {
    fprintf(stderr, "Bad fork - use chopsticks.\n");
    exit(1);
  }
  if (pid)
  {
    wait(&status);   /* synchronize to wait for child process to end */
    system("date");  /* the argument gets executed as if it was typed in */
  }
  else
    printf("This was printed out on ");
}
```

```
thor> gcc fork2.c
thor> a.out
Hello
This was printed out on Mon Feb 15 13:47:53 EST 1993
thor>
```

example of alarm and kill functions

```
/*
 *  killer.c  -  John K. Estell  -  28 February 1996
 *  example of using alarms and passing signals to another process
 */

#include <stdio.h>
#include <signal.h>
#include <sys/wait.h>

int pid;

int myalarm(void);

main()
{
   int status;

   pid = fork();
   if (pid == -1)
      exit(1);    /* we’ll spare you another fork joke... */

   if (pid == 0) /* child process */
      while (1)
      {
         printf("I'm a child process running wild and free!!!\n");
         sleep(1);
      }
   else /* parent process */
   {
      signal(SIGALRM, myalarm);
      alarm(5);                  /* alarm will go off in 5 seconds... */
      printf("Kid, you're annoying me....\n");
      wait(&status);
      printf("Ahhh.... MUCH better!!!\n");
   }
}

int myalarm(void)
{
   kill(pid, SIGKILL);  /* send the terminate process signal to the child */
   printf("Get lost kid!!!\n");
}
```

```
thor> killer
I'm a child process running wild and free!!!
Kid, you're annoying me....
I'm a child process running wild and free!!!
I'm a child process running wild and free!!!
I'm a child process running wild and free!!!
I'm a child process running wild and free!!!
Get lost kid!!!
Ahhh.... MUCH better!!!
```

```
/*
 *  pipe0.c  -  John K. Estell  -  15 February 1993
 *  pipe example - communicating between processes
 */

#include <stdio.h>
#include <sys/file.h>
#include <sys/wait.h>

main()
{
  int fileids[2];   /* file descriptors for the pipe:         */
                    /* fileids[0] refers to read end of pipe  */
                    /* fileids[1] refers to write end of pipe */

  int pid;          /* process id for the fork */
  char buffer[80];
  int status;

  /* create pipe */

  if (pipe(fileids) == -1)
  {
    fprintf(stderr, "Error creating pipe - call a plumber\n");
    exit(1);
  }

  /* fork the process */

  if (( pid = fork() ) == -1)
  {
    fprintf(stderr, "Bad fork - use three epees taped together\n");
    exit(1);
  }

  if (pid == 0)  /* child process */
  {
    close(fileids[1]);            /* close write end - child will read from pipe */
    read(fileids[0], buffer, 80); /* read at most 80 chars                       */
    printf("%s\n", buffer);       /* and print out the message                   */
    close(fileids[0]);            /* close read end of pipe -                    */
  }                               /*  when both ends are closed, pipe disappears */
  else  /* parent process */
  {
    close(fileids[0]);            /* close read end - parent will write to pipe */
    write(fileids[1], "Hello World!", 80);
    close(fileids[1]);            /* close write end of pipe...                 */
    wait(&status);                /* wait until child is finished else output   */
                                  /*  might appear AFTER the system prompt!     */
  }
}
```

```
/*
 *  pipe1.c  -  John K. Estell  -  15 February 1993
 *  pipe example - setting up connections between programs
 */

#include <stdio.h>

main()
{
  int fileids[2];   /* file descriptors for the pipe */
                    /* fileids[0] refers to read end of pipe  */
                    /* fileids[1] refers to write end of pipe */

  int pid;          /* process id for the fork */

  /* create the pipe */

  if (pipe(fileids) == -1)
  {
    fprintf(stderr, "Error in creating pipe.\n");
    exit(1);
  }

  /* fork the process */

  pid = fork();
  if (pid == -1)
  {
    fprintf(stderr, "Bad fork - use a shovel.\n");
    exit(1);
  }

  if (pid == 0)   /* child process */
  {
    close(fileids[0]);    /* close read end - child is writing to the pipe    */
    dup2(fileids[1], 1);  /* map stdout to point to write end of pipe         */
    close(fileids[1]);    /* then close original end of duplicated file desc. */

   /* run the program - find who's logged in - pipe output to sort */

    execlp("who", "who", (char *) 0);
  }
  else   /* parent process */
  {
    close(fileids[1]);   /* close write end - parent is reading from the pipe */
    dup2(fileids[0],0);  /* map stdin to point to read end of pipe            */
    close(fileids[0]);   /* then close original end of duplicated file desc.  */

    /* run the program - sort data first column in ascending order -
       get input from the pipe                                        */

    execlp("sort", "sort", "+0", (char *) 0);
  }
}
```

```
jupiter> who
root     console Feb 10 09:16
anuthi   ttyp1   Feb 15 11:52   (cseserv)
kiraly   ttyp3   Feb 15 08:07   (micom)
gaustad  ttyp4   Feb 10 09:16   (zeus)
estell   ttyp5   Feb 15 13:57   (thor)
sbrack   ttyp6   Feb 15 12:44   (131.183.61.76)
bonehead ttypc   Feb 15 12:57   (missouri)
madimset ttypd   Feb 15 12:57   (cseserv)
marko    ttype   Feb 12 14:55   (hpwks)
jfeasby  ttyq6   Feb 15 13:09   (131.183.1.203)
jupiter> gcc pipe1.c
jupiter> a.out
anuthi   ttyp1   Feb 15 11:52   (cseserv)
bonehead ttypc   Feb 15 12:57   (missouri)
estell   ttyp5   Feb 15 13:57   (thor)
gaustad  ttyp4   Feb 10 09:16   (zeus)
jfeasby  ttyq6   Feb 15 13:09   (131.183.1.203)
kiraly   ttyp3   Feb 15 08:07   (micom)
madimset ttypd   Feb 15 12:57   (cseserv)
marko    ttype   Feb 12 14:55   (hpwks)
root     console Feb 10 09:16
sbrack   ttyp6   Feb 15 12:44   (131.183.61.76)
jupiter>
```

use of tempfiles for calling a program from within a program

```
/*
 *  sort1.c  -  John K. Estell  18 February 1993
 *  Sorting example using 'sort' program with temporary files
 *  to process data.  This routine will read in a person's name
 *  and his/her score; based on input this info will be sorted
 *  according to either last name or score.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define NUMSTUDENTS 10

main()
{
  int i, j, k, num, sel;
  int score[NUMSTUDENTS];
  char buf[80], lbuffer[132], *name[NUMSTUDENTS];
  char tmpfile1[20] = "/tmp/sort.XXXXXX";    /* temp file names - must have X's */
  char tmpfile2[25] = "/tmp/sort.outXXXXXX"; /* see docs on mktemp */
  FILE *tmp;

  /* get the number of entries */

  printf("How many entries: ");
  fgets(buf, sizeof(buf), stdin)
  sscanf(buf, "%d", &num);

  if (num > NUMSTUDENTS)
  {
    fprintf(stderr, "Too many entries - max is %d.\n", NUMSTUDENTS);
    exit(1);
  }

  /* get the data */

  for (i = 0; i < num; i++)
  {
    printf("Student #%d:\n", i+1);
    printf("Name (last, first) : ");
    fgets(buf, sizeof(buf), stdin);  /* get the actual input */
    for (j = 0; j < sizeof(buf); j++)
       if (buf[j] == '\n')  /* clobber the newline that fgets reads in */
          buf[j] = '\0';
    name[i] = (char *) malloc(strlen(buf)+1);
    strcpy(name[i], buf);
    printf("Score: ");
    fgets(buf, sizeof(buf), stdin);
    sscanf(buf, "%d", &score[i]);
  }
```

```
/* get the sorting preference selection */

do
{
  printf("Sort by name or by score?\n");
  printf("Enter 1 for name, 0 for score: ");
  fgets(buf, sizeof(buf), stdin);
  sscanf(buf, "%d", &sel);
} while ((sel != 0) && (sel != 1));


/*  create a temporary file for passing data to the sort routine;
 *  temp file will automatically be deleted upon termination of
 *  the process.
 */

mktemp(tmpfile1);        /* filenames are generated */
mktemp(tmpfile2);

/* open first temp file for writing unsorted data */

tmp = fopen(tmpfile1, "w");

/* write out the unsorted data */

for (i = 0; i < num; i++)
{
  fprintf(tmp, "%d\t%s\n", score[i], name[i]);
  free(name[i]);
}

/* close the file */

fclose(tmp);

/*  construct the appropriate command line for use as the argument to system
 *  e.g. "sort +1 -o /tmp/sort.outa12345  /tmp/sort.a12345" for alpha sort
 */

sprintf(buf, "sort +%d -o %s %s %s", sel, tmpfile2, sel ? " " : "-n", tmpfile1);

/* perform the sort */

system(buf);
```

```
  /* get the sorted data from the second temp file and process the results */

  i = 0;
  tmp = fopen(tmpfile2, "r");
  while (fgets(lbuffer, sizeof(lbuffer), tmp) != NULL)
  {
    sscanf(lbuffer, "%d", &score[i]);
    for (j = 0; j < sizeof(lbuffer); j++)
      if (isalpha(lbuffer[j]))
         break;
    for (k = j; k < sizeof(lbuffer); k++)
      if (!isprint(lbuffer[k]))
         break;
    name[i] = (char *) malloc(k-j+1);
    strncpy(name[i++], lbuffer+j, k-j);
  }
  fclose(tmp);

  /* delete the two temp files */

  unlink(tmpfile1);
  unlink(tmpfile2);

  /* output the results */

  for (i = 0; i < num; i++)
    printf("%s had a score of %d.\n", name[i], score[i]);
}
```

```
thor> gcc -o mysort sort1.c
thor> mysort
How many entries: 10
Student #1:
Name (last, first) : Zucchini, Zachary
Score: 87
Student #2:
Name (last, first) : Yucca, Yvonne
Score: 22
Student #3:
Name (last, first) : Watermelon, Walter
Score: 35
Student #4:
Name (last, first) : Tomato, Tommy
Score: 68
Student #5:
Name (last, first) : Squash, Samantha
Score: 87
Student #6:
Name (last, first) : Lettuce, Lisa
Score: 79
Student #7:
Name (last, first) : Rutabaga, Rita
Score: 96
Student #8:
Name (last, first) : Potato, Paul
Score: 100
Student #9:
Name (last, first) : Okra, Oscar
Score: 92
Student #10:
Name (last, first) : Nectarine, Natalie
Score: 86

Sort by name or by score?
Enter 1 for name, 0 for score: 1
Lettuce, Lisa had a score of 79.
Nectarine, Natalie had a score of 86.
Okra, Oscar had a score of 92.
Potato, Paul had a score of 100.
Rutabaga, Rita had a score of 96.
Squash, Samantha had a score of 87.
Tomato, Tommy had a score of 68.
Watermelon, Walter had a score of 35.
Yucca, Yvonne had a score of 22.
Zucchini, Zachary had a score of 87.
```

```
thor> mysort
How many entries: 10
Student #1:
Name (last, first) : Lettuce, Lisa
Score: 79
Student #2:
Name (last, first) : Nectarine, Natalie
Score: 86
Student #3:
Name (last, first) : Potato, Paul
Score: 100
Student #4:
Name (last, first) : Rutabaga, Rita
Score: 96
Student #5:
Name (last, first) : Yucca, Yvonne
Score: 22
Student #6:
Name (last, first) : Zucchini, Zachary
Score: 87
Student #7:
Name (last, first) : Squash, Samantha
Score: 87
Student #8:
Name (last, first) : Tomato, Tommy
Score: 68
Student #9:
Name (last, first) : Watermelon, Walter
Score: 35
Student #10:
Name (last, first) : Okra, Oscar
Score: 92

Sort by name or by score?
Enter 1 for name, 0 for score: 0
Yucca, Yvonne had a score of 22.
Watermelon, Walter had a score of 35.
Tomato, Tommy had a score of 68.
Lettuce, Lisa had a score of 79.
Nectarine, Natalie had a score of 86.
Squash, Samantha had a score of 87.
Zucchini, Zachary had a score of 87.
Okra, Oscar had a score of 92.
Rutabaga, Rita had a score of 96.
Potato, Paul had a score of 100.
thor>
```

```
/*
 *  sort2.c  -  John K. Estell  -  18 February 1993
 *  Sorting example using 'sort' program and pipes to process data.
 *  This routine will read in a person's name and his/her score;
 *  based on input this info will be sorted according to either
 *  last name or score
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <sys/wait.h>
#include <sys/file.h>

#define NUMSTUDENTS 10
#define LBUFLEN 132

main()
{
  int pipefd1[2], pipefd2[2];
  int pid1, pid2;
  int status;
  int i, j, k, num, select;
  int score[NUMSTUDENTS];
  char buffer[80], lbuffer[LBUFLEN], *name[NUMSTUDENTS];


  /* get the amount of data to be entered */

  printf("How many entries: ");
  fgets(buffer, sizeof(buffer), stdin);
  sscanf(buffer, "%d", &num);
  if (num > NUMSTUDENTS)
  {
    fprintf(stderr, "Too many entries - max is %d.\n", NUMSTUDENTS);
    exit(1);
  }

  /* get the student data */

  for (i=0; i<num; i++)
  {
    printf("Student #%d:\n", i+1);
    printf("Name (last, first) : ");
    fgets(buffer, sizeof(buffer), stdin);  /* get the actual input */
    buffer[strlen(buffer) - 1] = '\0';     /* clobber newline character... */
    name[i] = (char *) malloc(strlen(buffer)+1);
    strcpy(name[i], buffer);
    printf("Score: ");
    fgets(buffer, sizeof(buffer), stdin);
    sscanf(buffer, "%d", &score[i]);
  }
```

```
/* get sort preference */

 do
 {
   printf("Sort by name or by score?\n");
   printf("Enter 1 for name, 0 for score: ");
   fgets(buffer, sizeof(buffer), stdin);
   sscanf(buffer, "%d", &select);
 } while ((select != 0) && (select != 1));



 /* create the first pipe */

 if (pipe(pipefd1) == -1)
 {
   fprintf(stderr, "Error in creating pipe.\n");
   exit(1);
 }

 /* fork the first process */

 pid1 = fork();
 if (pid1 == -1)
 {
   fprintf(stderr, "Bad fork - use your fingers.\n");
   exit(1);
 }

 if (pid1 == 0)  /* *** ***** first child process ***** *** */
 {

   /* create pipe for second child */

   if (pipe(pipefd2) == -1)
   {
     fprintf(stderr, "Error in creating pipe.\n");
     exit(1);
   }

   /* fork the second child process */

   pid2 = fork();
   if (pid2 == -1)
   {
     fprintf(stderr, "Bad fork - use a straw.\n");
     exit(1);
   }
```

```
  if (pid2 == 0)  /* *** ***** second child process ***** *** */
  {

    /* write out stored data to pipe */

    close(pipefd2[0]);    /* close read end of pipe - not being used */
    for (i = 0; i < num; i++)
    {
      sprintf(buffer, "%d\t%s\n", score[i], name[i]);
      write(pipefd2[1], buffer, strlen(buffer));
      free(name[i]);  /* this memory is no longer needed... */
    }

    close(pipefd2[1]); /* close write end of pipe */
    exit(0);
  }


  else  /* *** ***** first child process again ***** *** */
  {

    /* set up pipes to read from stdin and write to stdout */

    close(pipefd2[1]);   /* close write end of pipe from second child */
    dup2(pipefd2[0], 0); /* map stdin to point to read end of this pipe */
    close(pipefd2[0]);   /* close original end of duplicated file descriptor */

    close(pipefd1[0]);   /* close read end of pipe to parent process */
    dup2(pipefd1[1], 1); /* map stdout to point to write end of this pipe */
    close(pipefd1[1]);   /* close original end of duplicated file descriptor */

    /* all input data is in the pipe - execute the sort routine */

    if (select)
      execlp("/bin/sort", "sort", "+1", (char *) 0);
    else
      execlp("/bin/sort", "sort", "+0", "-n", (char *) 0);

    /* if here then execlp failed.... */

    fprintf(stderr, "Error in running sort program.\n");
    exit(1);

  }
}
```

```
  else  /* *** ***** parent process ***** *** */
  {

    /* set up the pipe from the first child */

    close(pipefd1[1]);  /* close write end of pipe from first child */
    dup2(pipefd1[0],0); /* map stdin to the read end of the pipe */
    close(pipefd1[0]);  /* close duplicated file descriptor */

    /* wait for sort program to finish */

    wait(&status);

    /* read the output from the sort program */

    i = 0;
    while (gets(lbuffer) != NULL)
    {
      sscanf(lbuffer, "%d", &score[i]);     /* get the score */
      for (j = 0; j < LBUFLEN; j++)             /* look for the name */
        if (isalpha(lbuffer[j]))
           break;
      for (k = j; k < LBUFLEN; k++)             /* look for line termination */
        if (!isprint(lbuffer[k]))
           break;
      name[i] = (char *) malloc(k-j+1);     /* get and store the name */
      strncpy(name[i++], lbuffer+j, k-j+1);
    }

    /* output sorted results */

    for (i = 0; i < num; i++)
      printf("%s had a score of %d.\n", name[i], score[i]);
  }
}
```

```
thor> gcc -o mysort sort2.c
thor> mysort
How many entries: 20
Too many entries - max is 10.
thor> mysort
How many entries: 10
Student #1:
Name (last, first) : Illini-Sweetcorn, Irving
Score: 89
Student #2:
Name (last, first) : Flax, Freddie
Score: 100
Student #3:
Name (last, first) : Plum, Penny
Score: 98
Student #4:
Name (last, first) : Habanero, Herrietta
Score: 78
Student #5:
Name (last, first) : Cucumber, Christine
Score: 75
Student #6:
Name (last, first) : Eggplant, Ernie
Score: 58
Student #7:
Name (last, first) : Jalepeno, Jennifer
Score: 82
Student #8:
Name (last, first) : Banana, Barbara
Score: 98
Student #9:
Name (last, first) : Apricot, Alexander
Score: 92
Student #10:
Name (last, first) : Grapefruit, Gregory
Score: 86

Sort by name or by score?
Enter 1 for name, 0 for score: 1
Apricot, Alexander had a score of 92.
Banana, Barbara had a score of 98.
Cucumber, Christine had a score of 75.
Eggplant, Ernie had a score of 58.
Flax, Freddie had a score of 100.
Grapefruit, Gregory had a score of 86.
Habanero, Herrietta had a score of 78.
Illini-Sweetcorn, Irving had a score of 89.
Jalepeno, Jennifer had a score of 82.
Plum, Penny had a score of 98.
```

```
thor> mysort
How many entries: 10
Student #1:
Name (last, first) : Apricot, Alexander
Score: 92
Student #2:
Name (last, first) : Banana, Barbara
Score: 98
Student #3:
Name (last, first) : Cucumber, Christine
Score: 75
Student #4:
Name (last, first) : Plum, Penny
Score: 98
Student #5:
Name (last, first) : Eggplant, Ernie
Score: 58
Student #6:
Name (last, first) : Flax, Freddie
Score: 100
Student #7:
Name (last, first) : Grapefruit, Gregory
Score: 86
Student #8:
Name (last, first) : Habanero, Henrietta
Score: 78
Student #9:
Name (last, first) : Illini-Sweetcorn, Irving
Score: 89
Student #10:
Name (last, first) : Jalepeno, Jennifer
Score: 82

Sort by name or by score?
Enter 1 for name, 0 for score: 0
Eggplant, Ernie had a score of 58.
Cucumber, Christine had a score of 75.
Habanero, Henrietta had a score of 78.
Jalepeno, Jennifer had a score of 82.
Grapefruit, Gregory had a score of 86.
Illini-Sweetcorn, Irving had a score of 89.
Apricot, Alexander had a score of 92.
Banana, Barbara had a score of 98.
Plum, Penny had a score of 98.
Flax, Freddie had a score of 100.
thor>
```

```
/*
 *  getname.c  -  John K. Estell  -  9 March 1995
 *  source file for reusable functions: getting host and user name information.
 */

#include "getname.h"

/*getmyusername() - returns pointer to string with the user's login name. */

char *getmyusername(void)
{
   uid_t uid;
   struct passwd *p;

   uid = getuid();    /* get the uid of the user and use it to access the password entry */
   p = getpwuid(uid); /* information stored in static area - overwritten by next call */
   return p->pw_name; /* points to entry in the password structure... */
}

/*getmyhostname() - returns pointer to string containing the hostname. */

char *getmyhostname(void)
{
   static char machine_name[256];

   gethostname(machine_name, sizeof(machine_name));
   return machine_name;
}

/* getmyhostrecord() - returns pointer to record containing user's network address(es) */

struct hostent *getmyhostrecord(void)
{
   struct hostent *hp;  /* pointer to static area set up by system */

   hp = gethostbyname(getmyhostname()); /* get host name and network host entries... */
   if (hp == NULL)
   {
      perror("hostname lookup error");
      exit(1);
   }
   return hp;  /* note that subsequent calls will overwrite the static storage area... */
}

/* get my domainname() - returns pointer to string containing the domain name. */

char *getmydomainname(void)
{
   char domain[BUFSIZ];

   getdomainname(domain, BUFSIZ);
   return domain;
}
```

whereami: header file for reusable functions

```
/*
 *  getname.h  -  John K. Estell  -  9 March 1995
 *  header file for reusable functions: getting host and user name information
 */

#include <pwd.h>
#include <sys/types.h>
#include <netdb.h>
#include <string.h>

char *getmyusername(void);
char *getmyhostname(void);
char *getmydomainname(void);
struct hostent *getmyhostrecord(void);
```

To prepare reusable functions one creates a module that contains the source code for the reusable functions plus a header file that contains the function prototypes. The -c option of the compiler is used to create an object module from the source code module:

```
   gcc -c getname.c
```

When compiling a program that uses these function, one includes the header file for the resuable module, and includes the name of the object module when compiling. For example, the following would be entered if foo.c uses some of the functions contained in getname.c:

```
  gcc -o foo foo.c getname.o
```

```
/*
 *  whereami.c  -  John K. Estell  -  9 March 1995
 *  demo program for obtaining user and host information
 */

#include <stdio.h>
#include "getname.h"


main()
{
   int  user_id;
   int  i;
   int num_addresses = 0;
   struct hostent *myhost;

   myhost = getmyhostrecord();
   while (myhost->h_addr_list[num_addresses])
      num_addresses++;

   printf("You are %s.\n", getmyusername());
   printf("You are on %s.\n", getmyhostname());
   printf("Your domain is %s.\n", getmydomainname());
   if (num_addresses == 1)
      printf("Its network address is: %s\n", inet_ntoa(myhost->h_addr_list[0]));
   else
   {
      printf("Its network addresses are:\n");
      i = 0;
      while (myhost->h_addr_list[i])
         printf("   %s\n", inet_ntoa(myhost->h_addr_list[i++]));
   }
}
```

```
thor> gcc -c getname.c
thor> gcc -o whereami whereami.c getname.o
thor> whereami
You are estell.
You are on thor.
Your domain is cse.utoledo.edu.
Its network address is: 131.183.56.3
thor> rlogin jupiter
{ a lot of login stuff deleted... }
jupiter> whereami
You are estell.
You are on jupiter.
Your domain is cse.utoledo.edu.
Its network address is: 131.183.51.23
jupiter> logout
```

Later on we'll look at the use of makefiles to automate the seperate compilation process.

IPC-1: connection-oriented UNIX domain client

```
/*
 *  client_unix.c  -  John K. Estell  -  22 July 1993
 *  Example of a client process used to obtain sum of two integers.
 *  Connection-oriented protocol, Unix domain
 *  SVR4: compile with libraries -lsocket -lnsl -lucb
 */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <string.h>

void report_fatal_error(char *message);


main()
{
   int    sd;                     /* socket descriptor */
   struct sockaddr_un sockname;   /* structure for socket name */
   int    i;

   int    sum;        /* application-specific declarations */
   struct mystruct
   {
      int addend;
      int augend;
   } data;

   /*
    *  create socket
    */

   sd = socket(AF_UNIX, SOCK_STREAM, 0);
   if (sd == -1) /* have error... */
      report_fatal_error("client: can not create socket");

   /*
    *  fill in Unix domain information
    */

   bzero((char *) &sockname, sizeof(sockname));
   sockname.sun_family = AF_UNIX;
   strcpy(sockname.sun_path, "/tmp/mysocket"); /* use any name for the temporary file */

   /*
    *  get the data to be sent
    */

   printf("Enter addend: ");
   scanf("%d", &data.addend);  /* doing it the quick way - assuming correct input! */
   printf("Enter augend: ");
   scanf("%d", &data.augend);
   /*
    *  connect to server
    */

   if (connect(sd, (struct sockaddr *) &sockname, sizeof(sockname)) == -1) /* error */
      report_fatal_error("bad connection");

   /*
    *  write to the socket: must specify file descriptor, address where the data to be
    *  transmitted starts, and the number of bytes to be sent
    */

   write(sd, &data, sizeof(data));    /* note that we are passing a structure.... */

   /*
    *  get response, then display the result
    */

   read(sd, &sum, sizeof(sum));
   printf("%d + %d = %d\n", data.addend, data.augend, sum);

   /*
    *  terminate connection
    */

   close(sd);
}


void report_fatal_error(char *message)
{
   perror(message);
   exit(1);
}
```

IPC-1: connection-oriented UNIX domain server

```c
/*
 *  server_unix.c  -  John K. Estell  -  22 July 1993
 *  example of a server program that adds two integers received from client
 *  Connection-oriented protocol, Unix domain, iterative server
 *  SVR4: compile with libraries -lsocket -lnsl -lucb
 */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <string.h>

main()
{
   int    sd;                      /* socket descriptor for local setup */
   int    new_sd;                  /* socket descriptor for connection */
   struct sockaddr_un serv_addr; /* used to describe local socket */
   int    sum;                     /* application-specific variables */
   struct mystruct
   {
      int addend;
      int augend;
   } values;

   void   report_fatal_error(char *message); /* used to exit program on error */

   /*
    *  create socket
    */

   sd = socket(AF_UNIX, SOCK_STREAM, 0);
   if (sd == -1)
      report_fatal_error("server: can't open stream socket");

   /*
    *  fill in Unix domain information
    */

   bzero((char *) &serv_addr, sizeof(serv_addr));
   serv_addr.sun_family = AF_UNIX;
   strcpy(serv_addr.sun_path, "/tmp/mysocket");

   /*
    *  register socket name
    */

   if (bind(sd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) == -1)
      report_fatal_error("server: binding error in local address");
   /*
    *  allow clients access to server
    */

   listen(sd, 5);

   /*
    *  loop forever - iterative server - wait for client access
    */

   while (1)
   {
      new_sd = accept(sd, NULL, 0);  /* use NULL and 0 if not interested in foreign host */
      if (new_sd == -1)
         report_fatal_error("server: accept error");

      /*
       *  get data sent from client
       */

      read(new_sd, &values, sizeof(values));   /* receiving a structure.... */
      sum = values.addend + values.augend;
      printf("server: %d + %d = %d\n", values.addend, values.augend, sum); /* demo only! */

      /*
       *  send sum back to client
       */

      write(new_sd, &sum, sizeof(sum));    /* writing an integer... */

      /*
       *  close connection
       */

      close(new_sd);
   }
}


void report_fatal_error(char *message)
{
   perror(message);
   exit(1);
}
```

```
thor> server_unix &
[1] 3904
thor> client_unix
Enter addend: 2
Enter augend: 3
server: 2 + 3 = 5
2 + 3 = 5
```

IPC-2: connection-oriented Internet domain client

```
/*
 *  client_inet.c  -  John K. Estell  -  19 July 1993
 *  example of a client process used to write a string to a server
 *  Connection-oriented protocol, Internet domain
 *  SVR4: compile with libraries -lsocket -lnsl -lucb
 */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>

void   report_fatal_error(char *message);

main()
{
   char   buf[256];
   int    sd;                      /* socket descriptor         */
   struct sockaddr_in sockname;    /* structure for socket name */
   int    i;

   /*
    *  create socket
    */

   sd = socket(AF_INET, SOCK_STREAM, 0);
   if (sd == -1)
      report_fatal_error("client: can't open socket");

   /*
    *  fill in Internet domain information
    */

   bzero((char *) &sockname, sizeof(sockname));
   sockname.sin_family      = AF_INET;
   sockname.sin_addr.s_addr = inet_addr("131.183.56.3");
   sockname.sin_port        = htons(9876);


   /*
    *  connect to server
    */

   if (connect(sd, (struct sockaddr *) &sockname, sizeof(sockname)) == -1)
      report_fatal_error("client: bad connection");
   /*
    *  get the data to be sent
    */

   printf("Enter string: ");
   fgets(buf, sizeof(buf), stdin);

   /*
    *  write to the socket
    */

   write(sd, buf, strlen(buf)+1);
   close(sd);
}


void report_fatal_error(char *message)
{
   perror(message);
   exit(1);
}
```

IPC-2: connection-oriented Internet domain server

```
/*
 *  server_inet.c  -  John K. Estell  -  19 July 1993
 *  Example of a server program that receives, then displays, string from client
 *  Connection-oriented protocol, Internet domain, iterative server
 *  SVR4: compile with libraries -lsocket -lnsl -lucb
 */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

void   report_fatal_error(char *message); /* used to exit program on error */

main()
{
   int    sd;                     /* socket descriptor for local setup         */
   int    new_sd;                 /* socket descriptor for connection          */
   char   buf[256];               /* data buffer for passed string             */
   struct sockaddr_in serv_addr; /* used to describe local socket             */
                                  /* {protocol, local-addr, local-process}     */
   struct sockaddr_in peer_addr; /* stores info on foreign socket             */
                                  /* {protocol, foreign-addr, foreign-process} */
   int    peer_len;               /* size of peer_addr structure - must be set */
   int    i;                      /* generic                                   */

   /*
    *  create socket
    */

   sd = socket(AF_INET , SOCK_STREAM, 0);
   if (sd == -1)
      report_fatal_error("server: can't open stream socket");

   /*
    *  fill in Internet domain information
    */

   bzero((char *) &serv_addr, sizeof(serv_addr));
   serv_addr.sin_family      = AF_INET;
   serv_addr.sin_addr.s_addr = inet_addr("131.183.56.3");
   serv_addr.sin_port        = htons(9876);

   /*
    *  register socket name
    */

   if (bind(sd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) == -1)
      report_fatal_error("server: binding error in local address");
   /*
    *  allow clients access to server
    */

   listen(sd, 5);


   /*
    *  loop forever - wait for client access
    */

   while (1)
   {
      peer_len = sizeof(peer_addr);  /* must be set to get peer's address */
      new_sd = accept(sd, (struct sockaddr *) &peer_addr, &peer_len);
      if (new_sd == -1)
         report_fatal_error("server: accept error");

      /*
       *  get data sent from client
       */

      i = read(new_sd, buf, sizeof(buf));
      printf("server received \"%s\" ", buf);
      printf("from host %s\n", inet_ntoa(peer_addr.sin_addr));
      printf("it was %d bytes long\n", i);
      close(new_sd);
   }
}


void report_fatal_error(char *message)
{
   perror(message);
   exit(1);
}
```

```
thor> server_inet &
[2] 3909
thor> rlogin jupiter
Password:

jupiter> client_inet
Enter string: This is a test.
server received "This is a test." from host 131.183.56.1
it was 16 bytes long
jupiter> logout
Connection closed.
thor> client_inet
Enter string: Howdy doody
server received "Howdy doody" from host 131.183.56.3
it was 12 bytes long
thor>
```

cseclass: client program for accessing course information

```
/*
 *  cseclass.c  -  John K. Estell  -  11 December 1995
 *  client program for cseclassd
 *  Allows entry of cse course number, will return name of
 *  professor teaching course, meeting time, and room if course
 *  is offered - or inform user that course is not offered.
 *
 *  usage: cseclass [hostname]
 *  if no host specified, will assume a default host as set in header file
 *  SVR4: compile with libraries -lsocket -lnsl -lucb
 */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <string.h>
#include <netinet/in.h>
#include <netdb.h>

#include "cseclass.h"   /* #define statements common to both
                           client and server                 */

struct hostent *get_host_pointer(int num_args, char **arg_list);
void make_connection(int sd, struct sockaddr_in *serv_addr, struct hostent *hp);
void print_usage_info(void);
char *get_string_input(char *prompt, char *buffer, int buffer_length);
void disconnect_socket(int sd);


main (int argc, char *argv[])
{
   int    sd;                         /* socket descriptor */
   struct sockaddr_in  serv_addr;
   struct hostent     *hp;            /*  Host entry  */
   char inbuf[10], sendbuf[4];
   COURSEREC course_data;             /* record used for storing data */


   sd = socket(AF_INET, SOCK_STREAM, 0);  /* create Internet stream socket */
   if (sd == -1) /* couldn't create socket - exit program */
   {
      perror(argv[0]);
      exit(1);
   }

   hp = get_host_pointer(argc, argv);     /* get host entry pointer */

   make_connection(sd, &serv_addr, hp);   /* connect to server */

   print_usage_info();
   while (get_string_input("course number> ", inbuf, sizeof(inbuf)) != NULL)
   {
      sscanf(inbuf, "%s", sendbuf);          /* send request to server */
      write(sd, sendbuf, 1+strlen(sendbuf));
      bzero(&course_data, sizeof(course_data));  /* zero out storage area */
      read(sd, inbuf, OFFER_SIZE);   /* wait for response from server */

      if (!strcmp(inbuf, IS_OFFERED))  /* check to see if course is offered */
      {
         read(sd, &course_data, sizeof(COURSEREC)); /* get course info */
         printf("CSE %d will be taught by Professor %s.\n",
                course_data.number, course_data.profname);
         printf("It meets in room %s.\n", course_data.roomloc);
         printf("Class lectures are scheduled for %s\n", course_data.meeting);
      }
      else  /* course is not offered */
         printf("CSE %s is not being offered.\n", sendbuf);
   }

   disconnect_socket(sd);  /* tell server that we're done */
}

/*** functions  ***/

struct hostent *get_host_pointer(int num_args, char **arg_list)
{
   struct hostent *hp;

   switch (num_args)
   {
      case 1:
              hp = (struct hostent *) gethostbyname(DEFAULT_HOST);
              if (hp == 0)
              {
                 fprintf(stderr, "Cannot resolve %s.\n", DEFAULT_HOST);
                 exit(1);
              }
              break;
      case 2:
              hp = (struct hostent *) gethostbyname(arg_list[1]);
              if (hp == 0)
              {
                 fprintf(stderr, "Cannot resolve %s.\n", arg_list[1]);
                 exit(1);
              }
              break;
      default:
              fprintf(stderr, "Usage: %s [hostname]\n", arg_list[0]);
              exit(1);
   }

   return hp;
}
```

```
/*
 *  make_connection():
 *  connect to the server's socket - must specify server's socket name
 *  this will cause the server to spawn a child process to handle any
 *  requests made by this client
 */

void make_connection(int sd, struct sockaddr_in *serv_addr, struct hostent *hp)
{
   serv_addr->sin_family = AF_INET;
   bcopy(hp->h_addr, &serv_addr->sin_addr, hp->h_length);
   serv_addr->sin_port = htons(PORT_NUM);

   if (connect(sd, serv_addr, sizeof(struct sockaddr_in)) == -1)
   {
      perror("Bad connection in cseclass");  /* no connection to daemon */
      close(sd);
      exit(1);
   }
}

void print_usage_info(void)
{
   printf("CSE Class Offerings\n");
   printf("Enter course number (last 3 digits only) to get info\n");
   printf("Press return at prompt to quit\n");
}

char *get_string_input(char *prompt, char *buffer, int buffer_length)
{
   int i = 0;

   printf("%s", prompt);
   fgets(buffer, buffer_length, stdin);
   while (buffer[i] != ’\0’)
   {
      if (buffer[i] == '\n') /* clobber the newline that appears before the NUL char. */
      {
         buffer[i] = 0;
         break;
       }
      i++;
   }
   if (i <= 1)   /* either "" or "\n" */
      return NULL;
   else
      return buffer;
}

void disconnect_socket(int sd)
{
   write(sd, END_CHILD_PROCESS, END_SIZE);
   close(sd);
}
```

```
/*
 *  cseclassd.c  -  John K. Estell  -  11 December 1995
 *  server program for cseclass
 *  This is a daemon process, to be run in the background with
 *  the possibility of multiple connections to the server.
 *  For each client connection a child process is created to
 *  handle the request.
 *  NOTE: this is not the proper way of handling the forking of
 *  service processes, as this program will generate zombies
 *  After this illustrative example, the proper way will be shown.
 *  SVR4: compile with libraries -lsocket -lnsl -lucb
 */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <signal.h>
#include <netinet/in.h>
#include <netdb.h>
#include "cseclass.h"   /* #define statements common to both
                            client and server                 */

/***
 ***   GLOBALS
 ***/

int     sd;              /* socket descriptor */
int     ns;
int     num_courses;
FILE    *fd;
struct  sockaddr_in serv_addr;
struct  sockaddr_in peer_addr;
int     peer_len;

COURSEREC *course;       /* storage for courses will be dynamically allocated */


/***
 ***  FUNCTION PROTOTYPES
 ***/


int   onintr(void);               /* signal handler used to exit server */
char *remove_cr(char *string);    /* remove <CR> from fgets-obtained string */
void  get_entry(int i);           /* gets one record of course data */
void  fatal_error(char *string);  /* exits program on fatal error */
void  child_process(void);        /* handles child process */
```

```
/***
 ***
 ***  MAIN
 ***
 ***/

main()
{
   int  i;
   int  pid;
   char buf[BUFSIZE];

   /*
    *  open course data file - read in all data now before serving clients
    */

   if ((fd = fopen(DATA_FILE, "r")) == NULL)
   {
      fprintf(stderr, "Course offering file not found.\n");
      exit(1);
   }
   else
   {
      fgets(buf, BUFSIZE, fd);     /* get number of courses, then allocate space */
      sscanf(buf, "%d", &num_courses);
      course = (COURSEREC *) malloc(num_courses * sizeof(COURSEREC));

      for (i=0; i<num_courses; i++)     /* get course data */
         get_entry(i);

      fclose(fd);   /* close the data file */
   }

   sd = socket(AF_INET, SOCK_STREAM, 0);  /* create connection-oriented Internet socket */
   if (sd == -1) /* have error */
      fatal_error("Can't create socket");

   /*
    *  register the socket name
    *  allow connections using any Internet interface to the host machine.
    *  host port number specified in header file.
    */

   serv_addr.sin_family      = AF_INET;
   serv_addr.sin_addr.s_addr = INADDR_ANY;    /* allow connection from any foreign host */
   serv_addr.sin_port        = htons(PORT_NUM);

   if (bind(sd, &serv_addr, sizeof(serv_addr)) == -1)
      fatal_error("Bad binding in cseclassserver");
   /*
    *  redirect various interrupts so that, when used, socket will be removed
    */

   signal(SIGINT, onintr);
   signal(SIGHUP, onintr);
   signal(SIGSTOP,onintr);
   signal(SIGTSTP,onintr);
   signal(SIGQUIT,onintr);
   signal(SIGTERM,onintr);

   /*
    *  allow up to five users to await a connection at any one time
    */

   listen(sd,5);
   peer_len = sizeof(peer_addr); /* no need for this to be in the loop... */

   /***
    ***  daemon loop
    ***/

   while (1)
   {
      /*
       *  wait for a client to connect for service - after accept() one would check
       *  information in peer_addr structure if access is to be restricted to a set
       *  of foreign hosts.
       */


      ns = accept(sd, &peer_addr, &peer_len);

      /*
       *  fork a child to provide required service
       */

      pid = fork();
      if (pid == -1)
         fatal_error("unable to fork server process");

      if (pid == 0)
         child_process(); /* use child process to serve client request */
      else
         close(ns);       /* not needed by parent */
   }
}
```

```
/***
 ***  FUNCTIONS
 ***/

/*
 *  child_process: handles the child process
 */

void child_process(void)
{
   int  course_request;
   int  flag;
   int  i;
   char buf[BUFSIZE];

   close(sd);  /* not needed by child */
   while(1)   /* handle requests from client until "END" is sent */
   {
      bzero(buf, BUFSIZE);
      read(ns, buf, BUFSIZE);
      if (strcmp(buf, END_CHILD_PROCESS) == 0)
         break; /* exit while loop */

      flag = 0;
      sscanf(buf, "%d", &course_request); /* get course number requested */

      /* check to see if the course is offered by scanning the course list */

      for(i=0; i<num_courses; i++)
         if (course[i].number == course_request) /* tell client course is offered,
                                                    then send the data             */
         {
            write(ns, IS_OFFERED, OFFER_SIZE);
            write(ns, &course[i], sizeof(COURSEREC));
            flag = 1;
         }

      if (!flag)  /* course is not offered */
         write(ns, IS_NOT_OFFERED, OFFER_SIZE);
   }

   close(ns); /* finished with client - close the new socket */
   exit(0);
}

/*
 *  onintr: signal handler routine - used to eliminate socket
 */

int onintr()
{
   close(sd);
   exit(0);
}
```

```
/*
 *  remove_cr: removes annoying newline from strings read by fgets
 */

char *remove_cr(char *string)
{
   int i=0;
   while(string[i])
   {
      if (string[i] == '\n')
         string[i] = '\0';
      i++;
   }
   return string;
}


/*
 *  get_entry: reads course entry from data file
 */

void get_entry(int i)
{
   char buf[BUFSIZE];
   fgets(buf, BUFSIZE, fd);                          /* get course number */
   sscanf(buf, "%d", &course[i].number);
   fgets(buf, BUFSIZE, fd);                          /* get professor name */
   strcpy(course[i].profname, remove_cr(buf));
   fgets(buf, BUFSIZE, fd);                          /* get room location */
   strcpy(course[i].roomloc, remove_cr(buf));
   fgets(buf, BUFSIZE, fd);                          /* get meeting time */
   strcpy(course[i].meeting, remove_cr(buf));
}


/*
 *  fatal_error: report error and exit program
 */

void fatal_error(char *message)
{
   fprintf(stderr, "%s\n", message);
   perror();
   exit(1);
}
```

cseclass: header file for common constants and structures

```
/*
 *  cseclass.h  -  John K. Estell  -  11 December 1995
 *  header file for cseclass
 *
 *  Port number for socket used by non-root user must be greater than 1023;
 *  preferably, it should be greater than 5000 and must be less than 65536.
 *  This port number must be unique to your application - it cannot be used
 *  by other applications.
 */

#define PORT_NUM           6801
#define BUFSIZE            256
#define IS_OFFERED         "Y"
#define IS_NOT_OFFERED     "N"
#define OFFER_SIZE         2
#define END_CHILD_PROCESS  "END"
#define END_SIZE           4
#define DATA_FILE          "cse_offered_courses"
#define DEFAULT_HOST       "jupiter.cse.utoledo.edu"

/* define structure for course records */

typedef struct
        {
           int  number;        /* course number */
           char profname[30];  /* name of professor teaching course */
           char roomloc[10];   /* location of lecture room */
           char meeting[20];   /* day and time of lecture */
        } COURSEREC;
```

```
4
302
Kermit D. Frog
4150 SE
MTRF 06:00-07:40
305
Kitty Ann Litter
2935 UH
TRF 08:00-08:50
416
Jill E. Phish
1231 SM
MTWRF 12:00-13:50
418
Bill D. Katt
1200 ES
TR 17:35-19:15
```

cseclass: example of zombie processes

```
{ cseclassd has been compiled on jupiter; cseclass has been compiled on thor
  and (after modification for cc compiler) on maine.                         }

thor> rlogin jupiter  { deleted all the login stuff that normally follows... }

jupiter> cseclassd &    { start up the daemon }
[1] 16575
jupiter> ps
  PID TT STAT  TIME COMMAND
16551 p6 S     0:00 -csh (csh)
16575 p6 S     0:00 cseclassd    { daemon process is running }
16578 p6 R     0:00 ps

jupiter> logout      { daemon process continues to run..... }
Connection closed.
thor> cseclass
CSE Class Offerings
Enter course number (last 3 digits only) to get info
Press return at prompt to quit
CSE course number> 416
CSE 416 will be taught by Professor Jill E. Phish.
It meets in room 1231 SM.
Class lectures are scheduled for MTWRF 12:00-13:50
CSE course number> 300
CSE 300 is not being offered.
CSE course number>
thor> cseclass uoftcse  { no daemon is operating on uoftcse.... }
Bad connection in cseclass: Connection refused
thor> rlogin maine  { run the client from "farther away"; deleted login stuff }
maine> cseclass jupiter
CSE Class Offerings
Enter course number (last 3 digits only) to get info
Press return at prompt to quit
CSE course number> 302
CSE 302 will be taught by Professor Kermit D. Frog.
It meets in room 4150 SE.
Class lectures are scheduled for MTRF 06:00-07:40
CSE course number>
maine> logout
Connection closed.
thor> rlogin jupiter   { again, deleted the login stuff.... }
jupiter> ps -x
  PID TT STAT  TIME COMMAND   { ps -x will show all of your processes }
16575 ?  I     0:00 cseclassd
16579 ?  Z     0:00 <defunct>
16581 ?  Z     0:00 <defunct>
16583 p8 S     0:00 -csh (csh)
16605 p8 R     0:00 ps -x
jupiter> logout
```

{One problem that we need to address is the handling of zombie processees. Zombies are caused by the parent not waiting to receive the termination status of the forked process. However, if the daemon waits for the service process, it can't truly act as a daemon! The next example shows how to properly write the daemon program.}

```
/*
 *  cseclassd.c  -  John K. Estell  -  11 December 1995
 *  server program for cseclass
 *  This is a daemon process, to be run in the background with
 *  the possibility of multiple connections to the server.
 *  For each client connection a child process is created to
 *  handle the request.  This version properly handles the
 *  problem of the service processes becoming zombies through use
 *  of the double fork technique, which causes the service process
 *  to be inherited by init.
 *  SVR4: compile with libraries -lsocket -lnsl -lucb
 */

#include <stdio.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/socket.h>
#include <signal.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <netdb.h>
#include "cseclass.h"   /* #define statements common to both
                            client and server                 */

/***
 ***   GLOBALS
 ***/

int     sd;              /* socket descriptor */
int     ns;
int     num_courses;
FILE    *fd;
struct  sockaddr_in serv_addr;
struct  sockaddr_in peer_addr;
int     peer_len;

COURSEREC *course;       /* storage for courses will be dynamically allocated */


/***
 ***  FUNCTION PROTOTYPES
 ***/


int   onintr(void);               /* signal handler used to exit server */
char *remove_cr(char *string);    /* remove <CR> from fgets-obtained string */
void  get_entry(int i);           /* gets one record of course data */
void  fatal_error(char *string);  /* exits program on fatal error */
void  child_process(void);        /* handles child process */
void  get_data(char *data_file);  /* handles reading in of data from file */
```

```
/***
 ***
 ***  MAIN
 ***
 ***/

main()
{
   int  i;
   pid_t pid_1, pid_2;
   char buf[BUFSIZE];
   int status;
   int flag = 1;


   get_data(DATA_FILE);  /* handles reading in of data from file */

   sd = socket(AF_INET, SOCK_STREAM, 0);  /* create socket */
   if (sd < 0)
      fatal_error("Can’t create socket");

   /*
    *  register the socket name
    *  allow connections using any Internet interface to the host machine.
    *  host port number specified in header file.
    */

   serv_addr.sin_family      = AF_INET;
   serv_addr.sin_addr.s_addr = INADDR_ANY;
   serv_addr.sin_port        = htons(PORT_NUM);

   if (bind(sd, &serv_addr, sizeof(serv_addr)) == -1)
      fatal_error("Bad binding in cseclassserver");

   /*
    *  redirect various interrupts so that, when used, socket will be removed
    */

   signal(SIGINT,  onintr);
   signal(SIGHUP,  onintr);
   signal(SIGSTOP, onintr);
   signal(SIGTSTP, onintr);
   signal(SIGQUIT, onintr);
   signal(SIGTERM, onintr);

   /*
    *  allow up to five users to await a connection at any one time
    */

   listen(sd, 5);
   /***
    ***  daemon loop
    ***/

   while (1)
   {
      /*
       *  wait for a client to connect for service
       */

      peer_len = sizeof(peer_addr);
      ns = accept(sd, &peer_addr, &peer_len);

      /*
       *  fork a child to provide required service
       */

      if ((pid_1 = fork()) == -1)
         fatal_error("unable to fork process");

      if (pid_1 == 0)  /* first child process */
      {
         if ((pid_2 = fork()) == -1)  /* fork the server process */
            fatal_error("unable to fork server process");
         if (pid_2 > 0)               /* first child process */
            exit(0);                  /* first child immediately exits... */
         else
            child_process();          /* use 2nd child process to serve client request */
      }
      else /* parent process */
      {
         if (waitpid(pid_1, NULL, 0) != pid_1)  /* wait for first child */
            fatal_error("waitpid error");
         close(ns);       /* not needed by parent */
      }
   }
}
```

```
/***
 ***  FUNCTIONS
 ***/

/* child_process: handles the child process */

void child_process(void)
{
   int  course_request;
   int  flag;
   int  i;
   char buf[BUFSIZE];

   close(sd);  /* not needed by child */

   /* handle requests from client until "END" is sent */

   while (1)
   {
      bzero(buf, BUFSIZE);
      read(ns, buf, BUFSIZE);
      if (strcmp(buf, END_CHILD_PROCESS) == 0)
         break;

      flag = 0;

      sscanf(buf, "%d", &course_request);            /* get the course number requested  */


      for (i = 0; i < num_courses; i++)          /* check to see if the course is offered */
         if (course[i].number == course_request)        /* tell client course is offered */
         {
            write(ns, IS_OFFERED, OFFER_SIZE);
            write(ns, &course[i], sizeof(COURSEREC));      /* sent client the course data */
            flag = 1;
         }

      if (!flag)                                             /* course is not offered */
         write(ns, IS_NOT_OFFERED, OFFER_SIZE);
   }

   close(ns);  /* finished with client - close the new socket */
   exit(0);
}
```

```
/*
 *  open course data file - read in all data now before serving clients
 */

void get_data(char *data_file)
{
   char buf[BUFSIZE];
   int i;

   if ((fd = fopen(data_file, "r")) == NULL)
      fatal_error("Course offering file not found.");
   else
   {
      /* get number of courses */

      fgets(buf, BUFSIZE, fd);
      sscanf(buf, "%d", &num_courses);

      /* dynamically allocate space for course data */

      course = (COURSEREC *) malloc(num_courses * sizeof(COURSEREC));

      /* get course data */

      for (i = 0; i < num_courses; i++)
         get_entry(i);

      /* close the data file */

      fclose(fd);
   }
}

/*
 *  onintr: signal handler routine - used to eliminate socket
 */

int onintr()
{
   close(sd);
   exit(0);
}
```

```
/*
 *  remove_cr: removes annoying newline from strings read by fgets
 */

char *remove_cr(char *string)
{
   int i=0;
   while(string[i])
   {
      if (string[i] == '\n')
         string[i] = '\0';
      i++;
   }
   return string;
}


/*
 *  get_entry: reads course entry from data file
 */

void get_entry(int i)
{
   char buf[BUFSIZE];
   fgets(buf, BUFSIZE, fd);                          /* get course number */
   sscanf(buf, "%d", &course[i].number);
   fgets(buf, BUFSIZE, fd);                          /* get professor name */
   strcpy(course[i].profname, remove_cr(buf));
   fgets(buf, BUFSIZE, fd);                          /* get room location */
   strcpy(course[i].roomloc, remove_cr(buf));
   fgets(buf, BUFSIZE, fd);                          /* get meeting time */
   strcpy(course[i].meeting, remove_cr(buf));
}


/*
 *  fatal_error: report error and exit program
 */

void fatal_error(char *message)
{
   fprintf(stderr, "%s\n", message);
   perror();
   exit(1);
}
```

```
thor> rlogin jupiter    {login info deleted...}
jupiter> ps -x
  PID TT STAT  TIME COMMAND
23159 pa S     0:00 -tcsh (tcsh)
23199 pa R     0:00 ps -x
jupiter> cseclassd &
[1] 23202
jupiter> ps -x
  PID TT STAT  TIME COMMAND
23159 pa S     0:00 -tcsh (tcsh)
23202 pa S     0:00 cseclassd
23203 pa R     0:00 ps -x

{at this time cseclass is executed from thor}

jupiter> ps -x
  PID TT STAT  TIME COMMAND
23159 pa S     0:00 -tcsh (tcsh)
23202 pa S     0:00 cseclassd
23206 pa S     0:00 cseclassd  {service process - 1st child process already completed}
23207 pa R     0:00 ps -x
jupiter> cseclass              {now there are two clients being served...}
CSE Class Offerings
Enter course number (last 3 digits only) to get info
Press return at prompt to quit
course number> 302
CSE 302 will be taught by Professor Kermit D. Frog.
It meets in room 4150 SE.
Class lectures are scheduled for MTRF 06:00-07:40
course number>
jupiter> ps -x
  PID TT STAT  TIME COMMAND
23159 pa S     0:00 -tcsh (tcsh)
23202 pa S     0:00 cseclassd
23206 pa S     0:00 cseclassd  {still handing requests from thor...}
23213 pa R     0:00 ps -x

{at this point the client on thor terminates}

jupiter> ps -x
  PID TT STAT  TIME COMMAND
23159 pa S     0:00 -tcsh (tcsh)
23202 pa S     0:00 cseclassd    {only the daemon is present - no zombies!}
23214 pa R     0:00 ps -x
jupiter> kill 23202
jupiter> logout
Connection closed.
thor>
```

```c
/*
 *  http.c  -  John K. Estell  -  5 March 1996
 *  client program for httpd; given a URL, this program will make
 *  an internet socket connection to the specified host and
 *  retrieve the specified file.
 *
 *  usage: http URL, where URL is of the form:
 *               http://hostname/pathname
 *  SVR4: compile with libraries -lsocket -lnsl -lucb
 */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <string.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>

#define  PORT_NUM     80
#define  BUFSIZE      256
#define  SERVICE      "http"

struct hostent *get_host_pointer(int num_args, char **arg_list);
void make_connection(int sd, struct sockaddr_in *serv_addr, struct hostent *hp);
void disconnect_socket(int sd);
void get_filename(char **arg_list, char *filename);

main (int argc, char *argv[])
{
   int    sd;                       /* socket descriptor */
   struct sockaddr_in  serv_addr;
   struct hostent     *hp;          /*  Host entry  */
   int bytes_read;
   char inbuf[10], sendbuf[4];
   char buf[BUFSIZE], filename[BUFSIZE];
   char ch;

   if ((sd = socket(AF_INET, SOCK_STREAM, 0)) < 0)  /* create Internet stream socket */
      exit(1); /* couldn't create the socket */

   hp = get_host_pointer(argc, argv);     /* get host entry pointer */
   get_filename(argv, filename);          /* get the filename to request */

   make_connection(sd, &serv_addr, hp);   /* connect to server */

   sprintf(buf, "GET %s HTTP/1.0\r\n\r\n", filename);  /* form request */
   write(sd, buf, strlen(buf));                        /* send request */

   while ((bytes_read = read(sd, &ch, sizeof(char))) > 0)  /* read and display result */
      write(STDOUT_FILENO, buf, bytes_read);

   disconnect_socket(sd);  /* tell server that we're done */
}
```

```
/*
 *  get_host_pointer: parses URL, finds the hostname, and sets up the host entry */
 */

struct hostent *get_host_pointer(int num_args, char **arg_list)
{
   struct hostent *hp;
   char buf[BUFSIZE];
   int found_host = 0;

   if (num_args != 2)
   {
      fprintf(stderr, "Usage: %s URL\n", arg_list[0]);
      exit(1);
   }

   found_host = sscanf(arg_list[1], "http://%[^/]s", buf);
   if (found_host == 0)
   {
      fprintf(stderr, "Cannot resolve URL %s.\n", arg_list[1]);
      exit(1);
   }
   else if (found_host == 1)
   {
      hp = (struct hostent *) gethostbyname(buf);
      if (hp == 0)
      {
         fprintf(stderr, "Cannot resolve host %s.\n", buf);
         exit(1);
      }
   }

   return hp;
}

/*
 *  make_connection: connect to the specified host using the specified well-known port
 */

void make_connection(int sd, struct sockaddr_in *serv_addr, struct hostent *hp)
{
   struct servent *sp;

   sp = getservbyname(SERVICE, (char *) 0); /* look up well-known port number */
   serv_addr->sin_family = AF_INET;
   bcopy(hp->h_addr, &serv_addr->sin_addr, hp->h_length);
   serv_addr->sin_port = htons(sp->s_port);
   if (connect(sd, serv_addr, sizeof(struct sockaddr_in)) == -1)
   {
      perror("Bad connection");  /* no connection to daemon */
      close(sd);
      exit(1);
   }
}
```

```
/*
 *  get_filename: parses filename out of provided URL
 */

void get_filename(char **arg_list, char *filename)
{
   char hostname[BUFSIZE], buf[BUFSIZE];

   sscanf(arg_list[1], "http://%[^/]s", hostname);
   sprintf(buf, "http://%s%%s", hostname);
   sscanf(arg_list[1], buf, filename);
}


void disconnect_socket(int sd)
{
   close(sd);
}
```

```
thor> http
Usage: http URL
thor> http nourlgiven
Cannot resolve URL nourlgiven.
thor> http http://unknownsite/
Cannot resolve host unknownsite.
thor> http http://cse.utoledo.edu/badname.html
HTTP/1.0 404 Not Found
Date: Tue, 05 Mar 1996 19:31:56 GMT
Server: NCSA/1.4
Content-type: text/html

<HEAD><TITLE>404 Not Found</TITLE></HEAD>
<BODY><H1>404 Not Found</H1>
The requested URL /badname.html was not found on this server.<P>
</BODY>
thor>
thor> http http://cse.utoledo.edu/home/whats_new.html
HTTP/1.0 200 Document follows
Date: Tue, 05 Mar 1996 19:31:38 GMT
Server: NCSA/1.4
Content-type: text/html
Last-modified: Wed, 28 Feb 1996 06:03:07 GMT
Content-length: 3084

<!DOCTYPE HTML SYSTEM "html.dtd">
<HTML><HEAD><TITLE>What's New in the World of EECS</TITLE></HEAD>
<H1> What's New in EECS </H1>
<HR>
<H3>Local Toledo Mud Hens Web Site Featured in Magazine.</H3>
<P>Recently the local <A HREF="http://www.cse.utoledo.edu/~MudHens"> Toledo Mud Hens Web
Page</A> (Ned Skeldon Stadium) was featured in the January/February 1996 issue of
<A HREF="http://www.umagazine.com">U. The National College Magazine</A>&#169, as part of
their article onpossible alternatives for spring break this year.  The page was created by
<A HREF="http://www.cse.utoledo.edu/~zoltan"> James Karocki</A> under the
supervision of <A HREF="http://www.cse.utoledo.edu/people/faculty/estell.html">
Dr. John Estell</A>.
<br>
CONGRATULATIONS JAMES!!!
<hr>
<H3>Dr. Estell's Reed Organ Home page has been rated in the Top 5%</H3>
<A HREF="http://www.pointcom.com"><IMG SRC="http://www.cse.utoledo.edu/images/
5percmdt.gif" ALIGN="LEFT" BORDER="0"></A>
<P><A HREF="http://www.cse.utoledo.edu/people/faculty/estell.html">Dr. John Estell</A>,
Assistant Professor in EECS, has just been recognized by the editors of Point
as one of the "Top 5% of the Web" for his
<A HREF="http://www.cse.utoledo.edu/~estell/organs/">Reed Organ Home Page</A>.
His page will now be listed with other great sites on Point, with the
review by their editors and a direct link to his site.
<BR>
Point is a free service which rates and reviews only the best, sharpest,
and funniest home pages on the World Wide Web.   Their ratings and reviews
have been featured on CNN and in many publications around the world,
including their new Macmillan book "The World Wide Web Top 1000."
<BR>
CONGRATULATIONS JOHN!!!
<HR>
<H3>Dr. Estell Receives Award</H3>
<P><A HREF="http://www.cse.utoledo.edu/people/faculty/estell.html">Dr. John Estell</A>
, Assistant Professor in EECS, was the recent recipient of
the John A. Curtis Lecture Award from the Computers in Education Division of
the American Society for Engineering Education.
<BR>
Dr. Estell was also recognized for his Reed Organ home page, by being 1 of
only 48 entries in the music category in the book, "World Wide Web Directory"
by Jamsa & Cope, which included over 8,000 sites.
<BR>
CONGRATULATIONS JOHN!!!
<HR>
<P>Last modified: <I>2/13/96</I></P>
</BODY></HTML>
```

```
/*
 *  main.c  -  John K. Estell  -  5 March 1996
 *  main module for smtp program - my mail sender!
 *  client program for smtpd; given an address, this program will make
 *  an internet socket connection to the specified host and
 *  mail a temporary file to the specified user.  If host not specified
 *  then domain of current host is used as the default.
 *
 *  usage: smtp user[@host]
 */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include "files.h"
#include "constants.h"
#include "user.h"
#include "params.h"
#include "mail.h"
#include "socket.h"


main (int argc, char *argv[])
{
   int    sd;                           /* socket descriptor */
   struct hostent *hp;                  /*  Host entry  */
   FILE *fp;
   char buf[BUFSIZE];                   /* generic buffer */
   char username[BUFSIZE];              /* name of user to send mail to */
   char myname[BUFSIZE];                /* my user name */
   char myrealname[BUFSIZE];            /* my full name */
   char tempfile[BUFSIZE];              /* temporary file for editing message */

   check_num_args(argc, 2, FATAL);      /* bail out if not exactly 2 arguments */

   /* get information on who the mail if from and who is it being sent to */

   get_username(argv[1], username);     /* get the receipient's username */
   get_my_name(myname);                 /* get my name and domain */
   get_my_realname(myrealname);         /* get my real name */

   /* set up a temporary file for the message to be sent */

   create_filename(tempfile, "/tmp/", "LS", "");
   fp = open_file(tempfile, "w", FATAL);
   write_header_info(fp, myname, myrealname, argv[1]);
   fclose(fp);
   /* use the vi editor to enter the message */

   sprintf(buf, "vi %s", tempfile);
   system(buf);

   /* set up our connection with the peer address */

   if ((sd = socket(AF_INET, SOCK_STREAM, 0)) < 0)  /* create Internet stream socket */
      report_fatal_error("Can’t create socket");
   hp = get_host_pointer(argv[1]);        /* get host entry pointer */
   make_connection(sd, hp);               /* connect to server */

   /* send the sendmail protocol, followed by the data, then end */

   smtp_protocol(sd, myname, username);
   send_message(sd, tempfile);
   end_message(sd);
   disconnect_socket(sd);                 /* tell server that we're done */
   unlink(tempfile);                      /* delete the temporary file */
}
```

smtp (sendmail): makefile

```
#
# makefile for the program smtp
#
# this program will send a mail message, using the sendmail protocol, to the
# specified user at the specified host.

# John K. Estell  5 March 1996
# Note: SVR4 systems must compile executable with libraries -lsocket -lnsl

CC = gcc
objects = main.o files.o constants.o user.o mail.o params.o socket.o

smtp:         $(objects)
              $(CC) $(objects) -o smtp -lsocket -lnsl

main.o:       constants.h files.h

user.o:       constants.h user.h

mail.o:       files.h constants.h mail.h

socket.o:     constants.h socket.h

params.o:     params.h constants.h

files.o:      files.h constants.h

constants.o:  constants.h
```

```
/*
 *  constants.h  --  John K. Estell  --  30 January 1996
 *
 *  list of generally used constants
 */


#ifndef TRUE
#define TRUE        (1 == 1)
#define FALSE       (1 == 0)
#endif

#ifndef FATAL
#define FATAL       1
#define NON_FATAL   0
#endif

#ifndef YES
#define YES         TRUE
#define NO          FALSE
#endif

#ifndef SET
#define SET         TRUE
#define CLEAR       FALSE
#endif

#ifndef BUFSIZE
#define BUFSIZE     1024
#endif

/* global constant */

char *thisprogram;   /* used to point to argv[0] */

/* function prototypes */

extern void report_fatal_error(/* char *string */);
```

```
/*
 *  socket.h  -  John K. Estell  -  5 March 1996
 *  socket functions module for smtp program - my mail sender!
 */

#define  SERVICE      "smtp"

struct hostent *get_host_pointer(/* char *address */);
void make_connection(/* int sd, struct hostent *hp */);
void disconnect_socket(/* int sd */);
```

```
/*
 *  files.h  --  John K. Estell  --  18 January 1996
 *
 *  Commonly-used routines for handling files and related functions.
 */

extern FILE *open_file(/* char *filename, char *access, int failure_mode */);
extern char *change_extension(/* char *oldfilename, char *newextension */);
extern void create_filename(/* char *target, char *dir,
                               char *prefix, char *suffix */);
extern void create_directory(/* char *target, char *prefix */);
```

```
/*
 *  mail.h  -  John K. Estell  -  5 March 1996
 *  mail function module for smtp program - my mail sender!
 */

void get_reply(/* int sd */);
void get_time_and_date(/* char *time_sent */);
void write_header_info(/* FILE *fp, char *from1, char *from2, char *to */);
void smtp_protocol(/* int sd, char *myname, char *sendername */);
void send_message(/* int sd, FILE *fp */);
void end_message(/* int sd */);
```

```
/*
 *  params.h  --  John K. Estell  --  24 January 1996
 *
 *  routines for handling common actions involving the parsing
 *  of command line parameters.
 */

extern int check_num_args(/* int actual, int expected, int failure_mode */);
extern char is_flag(/* char *parameter */);
```

```
/*
 *  user.h  -  John K. Estell  -  5 March 1996
 *  user function module for smtp program - my mail sender!
 */

void get_username(/* char *name, char *username */);
void get_my_name(/* char *myname */);
void get_my_realname(/* char *myrealname */);
void get_time_and_date(/* char *time_sent */);
```

```
/*
 *  constants.c  --  John K. Estell  --  31 January 1996
 *
 *  dedicated to functions that are constantly found in my programs....
 */

#include <stdio.h>
#include "constants.h"

/*
 *  report_fatal_error: prints out program name and our reason for
 *                      killing off this process.
 */

void report_fatal_error(char *string)
{
   fprintf(stderr, "%s: %s\n", thisprogram, string);
   exit(1);
}
```

- Note: for the sake of brevity, those functions in files.c not used by the smtp program have been edited out of this listing.

```
/*
 *  files.c  --  John K. Estell  --  18 January 1996
 *
 *  Commonly-used routines for handling files and functions related
 *  to files.
 *
 *  version 1.0 - 18 January 1996
 *
 */

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include "constants.h"
#include "files.h"

/* function prototypes */


/*
 *  open_file: opens specified file for specified access;
 *             incorporates error checking.  Can specify errors to
 *             be fatal or non-fatal through use of FATAL and NON_FATAL
 *             constants passed in failure_mode field.
 */

FILE *open_file(char *filename, char *access, int failure_mode)
{
   FILE *fp;

   fp = fopen(filename, access);
   if ((fp == NULL) && (failure_mode == FATAL))
      report_fatal_error("error opening file.");
   else
      return fp;
}
```

```
/*
 *  create_filename: will attempt to create unique filenames for
 *                   temporary files.  User has option of supplying
 *                   a prefix and a suffix; routine will supply a
 *                   unique pid-based number, which should suffice...
 *                   if the dir field specifies a path, then it must
 *                   include the final directory slash immediately
 *                   preceding the actual filename.
 */

void create_filename(char *target, char *dir, char *prefix, char *suffix)
{
   char buf[10];

   target[0] = '\0';
   if (strlen(dir) > 0) /* then copy the path */
      strcat(target, dir);
   if (strlen(prefix) > 0) /* then copy the prefix */
      strcat(target, prefix);
   sprintf(buf, "%d", getpid()); /* output the varying portion of the name */
   strcat(target, buf);
   if (strlen(suffix) > 0) /* then copy the suffix as an extension */
   {
      strcat(target, ".");
      strcat(target, suffix);
   }
}
```

```
/*
 *  mail.c  -  John K. Estell  -  5 March 1996
 *  mail function module for smtp program - my mail sender!
 *  contains functions related to the actual sending of mail.
 */

#include <stdio.h>
#include <string.h>
#include "files.h"
#include "constants.h"
#include "mail.h"


/*
 *  smtp_protocol: sends the sendmail commands along with the
 *                 appropriate parameters to the socket.
 *                 get_reply is used for synchronization.
 */

void smtp_protocol(int sd, char *myname, char *sendername)
{
   char buf[BUFSIZE];

   sprintf(buf, "mail from: %s\n", myname);
   write(sd, buf, strlen(buf));
   get_reply(sd);
   sprintf(buf, "rcpt to: %s\n", sendername);
   write(sd, buf, strlen(buf));
   get_reply(sd);
   sprintf(buf, "data\n");
   write(sd, buf, strlen(buf));
   get_reply(sd);
}



/*
 *  end_message: ends the sending of the message and tells the
 *               peer that we're done.
 */

void end_message(int sd)
{
   char buf[BUFSIZE];

   sprintf(buf, ".\n");
   write(sd, buf, strlen(buf));
   get_reply(sd);
   sprintf(buf, "quit\n");
   write(sd, buf, strlen(buf));
   get_reply(sd);
}
```

```
/*
 *  send_message: opens specified temporary file containing message
 *                and writes the contents to the socket.
 */

void send_message(int sd, char *tempfile)
{
   char buf[BUFSIZE];
   FILE *fp;

   fp = open_file(tempfile, "r", FATAL);
   while (1)
   {
      if (fgets(buf, BUFSIZE, fp) == NULL)
         break;
      if (strcmp(buf, ".\n") == 0)  /* avoid the wily hacker's attempts... */
         strcpy(buf, ". \n");
      if (strncmp(buf, "From ", 5) == 0) /* prevent mail readers' confusion */
         write(sd, ">", 1);
      write(sd, buf, strlen(buf));
   }
   fclose(fp);
}



/*
 *  write_header_info: writes out the Date:, From:, To:, and Subject:
 *                     fields into the temporary file.  User can fill
 *                     in the subject line when in the vi editor -
 *                     note that the other fields can also be edited
 *                     but no major harm can (should?) result. :-)
 */

void write_header_info(FILE *fp, char *from1, char *from2, char *to)
{
   char time_sent[BUFSIZE];

   get_time_and_date(time_sent);
   fprintf(fp, "Date: %s\n", time_sent);
   fprintf(fp, "From: %s (%s)\n", from1, from2);
   fprintf(fp, "To: %s\n", to);
   fprintf(fp, "Subject: \n\n");
}
```

```
/*
 *  get_reply: use to synchronize with peer connection; otherwise, we
 *             might send information before it's ready to receive it.
 *             Returned text is displayed; other things could just as
 *             easily be performed with this information.
 */

void get_reply(int sd)
{
   char charbuf[1024], strout[1024];

   read(sd, charbuf, sizeof(charbuf))
   sscanf(charbuf, "%[^\n]s", strout);  /* demo of how to handle newlines with sscanf */
   printf("%s\n", strout);
}
```

- Note: for the sake of brevity, those functions in params.c not used by the smtp program have been edited out of this listing.

```
/*
 *  params.c  --  John K. Estell  --  24 January 1996
 *
 *  routines for handling common actions involving the parsing
 *  of command line parameters.
 */

#include "params.h"
#include "constants.h"

/*
 *  check_num_args: compares actual number of arguments to an expected
 *                  number of arguments.  If fatal mode is used, program
 *                  will exit if the values don't agree.  In non-fatal
 *                  mode, routine will return 0 for equality, -1 for
 *                  fewer than expected arguments, and +1 for more than
 *                  expected arguments.
 */

int check_num_args(int actual, int expected, int failure_mode)
{
   if (failure_mode == FATAL)
   {
      if (actual == expected)
         return 0;
      if (actual < expected)
         report_fatal_error("too few arguments");
      if (actual > expected)
         report_fatal_error("too many arguments");
   }
   else /* non-fatal */
   {
      if (actual == expected)
         return 0;
      if (actual < expected)
         return -1;
      if (actual > expected)
         return 1;
   }
}
```

smtp (sendmail): source code for socket functions

```
/*
 *  socket.c  -  John K. Estell  -  5 March 1996
 *  socket functions module for smtp program - my mail sender!
 *  Contains functions specific for use with the socket mechanism.
 *  get_host_pointer is specific to the sendmail application;
 *  other functions are generic.
 */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <string.h>
#include <netinet/in.h>
#include <netdb.h>
#include "socket.h"
#include "constants.h"


/*
 *  get_host_pointer: gets the host information for the place we're
 *                    sending the mail to.  If no host is specified
 *                    then getdomainname() is used to get the local
 *                    hostname.
 */

struct hostent *get_host_pointer(char *address)
{
   struct hostent *hp;
   char buf[BUFSIZE], user[BUFSIZE], host[BUFSIZE];
   int found_host = 0;

   sscanf(address, "%[^@]s", user);   /* parse the address - first get user */
   strcpy(buf, user);
   strcat(buf, "@%s");                      /* form new search string */
   found_host = sscanf(address, buf, host); /* and now you get the host... */

   if (found_host < 1) /* then only the user was specified - assume local */
      getdomainname(host, BUFSIZE);

   /* get the information for the host */

   hp = (struct hostent *) gethostbyname(host);
   if (hp == 0)
   {
      fprintf(stderr, "Cannot resolve host %s.\n", host);
      exit(1);
   }

   return hp;
}
```

```
/*
 *  make_connection: make a connection to the specified host.
 *                   Will place the host data into the sockaddr_in structure
 *                   then make the actual connection.
 */

void make_connection(int sd, struct hostent *hp)
{
   struct servent *sp;
   struct sockaddr_in serv_addr;

   sp = getservbyname(SERVICE, (char *) 0);
   serv_addr.sin_family = AF_INET;
   bcopy(hp->h_addr, &serv_addr.sin_addr, hp->h_length);
   serv_addr.sin_port = htons(sp->s_port);

   if (connect(sd, &serv_addr, sizeof(struct sockaddr_in)) == -1)
   {
      perror("Bad connection");  /* no connection to daemon */
      close(sd);
      exit(1);
   }
}



/*
 *  disconnect_socket: exactly what it says.... closes down the socket.
 */

void disconnect_socket(int sd)
{
   close(sd);
}
```

smtp (sendmail): source code for user-identification functions

```c
/*
 *  user.c  -  John K. Estell  -  5 March 1996
 *  user function module for smtp program - my mail sender!
 *  Contains functions specific to obtaining user identification,
 *  and also the current time and date.
 */

#include <stdio.h>
#include <sys/types.h>
#include <string.h>
#include <pwd.h>
#include <time.h>
#include "constants.h"
#include "user.h"


/*
 *  get_my_name: uses uid to get password entry containing login name.
 */

void get_my_name(char *myname)
{
   uid_t uid;
   struct passwd *p;
   char domain[BUFSIZE];

   uid = getuid();
   p = getpwuid(uid);               /* p points to password information */
   strcpy(myname, p->pw_name);      /* get the login name */
   strcat(myname, "@");
   getdomainname(domain, BUFSIZE);  /* and add the domain to it */
   strcat(myname, domain);
}
```

```
/*
 *  get_my_realname: uses gecos field in password entry to get the user's
 *                   real name.
 */

void get_my_realname(char *myrealname)
{
   uid_t uid;
   struct passwd *p;

   uid = getuid();
   p = getpwuid(uid);               /* p points to password information */

   /*
    *  gecos field contains subfields separated by commas -
    *  first field is real name.  This code does not handle '@'
    *  expansion to login name if encountered in this field - sorry.
    */

   sscanf(p->pw_gecos, "%[^,]s", myrealname);
}



/*
 *  get_time_and_date: places current time and date into provided string.
 */

void get_time_and_date(char *time_sent)
{
   time_t current;

   time(&current);  /* get the current time */
   strftime(time_sent, BUFSIZE, "%a, %d %b %Y %H:%M:%S %Z", localtime(&current));
}



/*
 *  get_username: parses string "user@host" and extracts the username.
 */

void get_username(char *name, char *username)
{
   sscanf(name, "%[^@]s", username);
}
```

smtp (sendmail): demonstration of use and operation

```
thor> smtp estell@cse.utoledo.edu
```

- When this program is executed, the vi editor is invoked for obtaining the message:

```
Date: Thu, 07 Mar 1996 14:14:06 EST
From: estell@cse.utoledo.edu (John K. Estell)
To: estell@cse.utoledo.edu
Subject:

~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"/tmp/LS14696" 5 lines, 120 characters
```

- After we exit vi, the messages received from the sendmail server are displayed to the user. Note that we don't have to do this -- it's just interesting to see...

```
220 cse.utoledo.edu Sendmail 5.67/1.8 ready at Thu, 7 Mar 96 14:15:53 -0500
250 estell@cse.utoledo.edu... Sender ok
250 estell... Recipient ok
354 Enter mail, end with "." on a line by itself
250 Ok
thor> smtp estell        (Note that no host is specified)
220 cse.utoledo.edu Sendmail 5.67/1.8 ready at Thu, 7 Mar 96 14:16:30 -0500
250 estell@cse.utoledo.edu... Sender ok
250 estell... Recipient ok
354 Enter mail, end with "." on a line by itself
250 Ok
thor> smtp jestell@top.eng.utoledo.edu     (Note use of a foreign host)
220 top.eng.utoledo.edu Sendmail SMI-8.6/SMI-SVR4 ready at Thu, 7 Mar 1996 14:16:53 -0500
250 estell@cse.utoledo.edu... Sender ok
250 jestell... Recipient ok
354 Enter mail, end with "." on a line by itself
250 OAA11776 Message accepted for delivery
thor>
thor> smtp estell
```

- This time we decided to have some fun and edit the **`From:`** ***field; the screen below shows what the message looked like just prior to leaving vi:***

```
Date: Thu, 07 Mar 1996 14:17:42 EST
From: lholstein@moo.com (A bovine with an attitude)
To: estell
Subject: Forgery!

Moo.

~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"/tmp/LS14708" 5 lines, 104 characters
```

```
220 cse.utoledo.edu Sendmail 5.67/1.8 ready at Thu, 7 Mar 96 14:18:26 -0500
250 estell@cse.utoledo.edu... Sender ok
250 estell... Recipient ok
354 Enter mail, end with "." on a line by itself
250 Ok
thor>
thor> mail
Mail version SMI 4.0 Wed Oct 23 10:38:28 PDT 1991  Type ? for help.
"/usr/spool/mail/estell": 25 messages 3 new
   21 estell@cse.utoledo.edu Thu Mar  7 13:23   17/498   another possibility
   22 camavc@okway.okstate.edu Thu Mar  7 13:28   41/1425  Re: Making a Nameboard La
>N 23 estell@cse.utoledo.edu Thu Mar  7 14:15   22/559   This is a test
 N 24 estell@cse.utoledo.edu Thu Mar  7 14:16   15/407   test #2
 N 25 lholstein@moo.com  Thu Mar  7 14:18   15/359   Forgery!
& t23
Message 23:
From estell@cse.utoledo.edu Thu Mar  7 14:15:53 1996
Received: from thor by cse.utoledo.edu (5.67/1.8)
        id AA26480; Thu, 7 Mar 96 14:15:53 -0500
Message-Id: <9603071915.AA26480@cse.utoledo.edu>
Date: Thu, 07 Mar 1996 14:14:06 EST
From: estell@cse.utoledo.edu (John K. Estell)
To: estell@cse.utoledo.edu
Subject: This is a test
Content-Length: 189
X-Lines: 9
Status: RO

This is a test - this is *only* a test.

But it is a real nasty test
>From estell@cse.utoledo.edu

To end a mail message, one sends the following
.
That is, a period by itself on a line.


& t24
Message 24:
From estell@cse.utoledo.edu Thu Mar  7 14:16:30 1996
Received: from thor by cse.utoledo.edu (5.67/1.8)
        id AA26493; Thu, 7 Mar 96 14:16:30 -0500
Message-Id: <9603071916.AA26493@cse.utoledo.edu>
Date: Thu, 07 Mar 1996 14:16:07 EST
From: estell@cse.utoledo.edu (John K. Estell)
To: estell
Subject: test #2
Content-Length: 61
X-Lines: 2
Status: RO

Just showing that we can handle the default host situation.
```

For the attempted forgery, note that the top From line is accurate; however, it is the From: line in the body of the text that mail displays in its received mail list as to who sent the mail. Still, one should note that electronic mail can be forged and (unfortunately) it isn't hard for someone to do. Definitely a good reason for using digital signatures or other mechanisms for certifying that the mail you've received is authentic.

```
& t25
Message 25:
From estell@cse.utoledo.edu Thu Mar  7 14:18:27 1996
Received: from thor by cse.utoledo.edu (5.67/1.8)
        id AA26530; Thu, 7 Mar 96 14:18:26 -0500
Message-Id: <9603071918.AA26530@cse.utoledo.edu>
Date: Thu, 07 Mar 1996 14:17:42 EST
From: lholstein@moo.com (A bovine with an attitude)
To: estell
Subject: Forgery!
Content-Length: 6
X-Lines: 2
Status: RO

Moo.


& x
thor>
```

```
top> more mail.out
```
(This is the message that was sent to top - it was saved to a file)
```
From estell@cse.utoledo.edu Thu Mar  7 14:16:54 1996
Return-Path: <estell@cse.utoledo.edu>
Received: from thor.cse.utoledo.edu by top.eng.utoledo.edu (SMI-8.6/SMI-SVR4)
        id OAA11776; Thu, 7 Mar 1996 14:16:54 -0500
Message-Id: <199603071916.OAA11776@top.eng.utoledo.edu>
Date: Thu, 07 Mar 1996 14:16:59 EST
From: estell@cse.utoledo.edu (John K. Estell)
To: jestell@top.eng.utoledo.edu
Subject: The final test
Content-Length: 40

Can this get over to top?  I think so!


top>
```

```
#
# makefile for the program err2html
# John K. Estell  30 January 1996
# The "err2html" program development tool.
#
# This tool takes a C source code file and runs it through both
# the gcc compiler and the lint program verifier.  The output is
# in the form of several HTML files, with one file containing the
# source code of the program and the remaining files containing
# the error message(s) for each line in the source file that has
# an error.  The error messages are accessed from the HTML source
# code file, where each line that is reported to contain an error
# is linked to the corresponding HTML error file.
#

CC = gcc
objects = main.o html.o files.o params.o constants.o

err2html:     $(objects)
              $(CC) $(objects) -o err2html

main.o:       main.h constants.h

html.o:       html.h files.h constants.h

files.o:      files.h constants.h

params.o:     params.h constants.h

constants.o:  constants.h
```

```
/*
 *  constants.h  --  John K. Estell  --  30 January 1996
 *
 *  list of generally used constants
 */

#include <stdio.h>

#ifndef TRUE
#define TRUE        (1 == 1)
#define FALSE       (1 == 0)
#endif

#ifndef FATAL
#define FATAL       1
#define NON_FATAL   0
#endif

#ifndef YES
#define YES         TRUE
#define NO          FALSE
#endif

#ifndef SET
#define SET         TRUE
#define CLEAR       FALSE
#endif

/* global constant */

char *thisprogram;   /* used to point to argv[0] */

/* function prototypes */

extern void report_fatal_error(/* char *string */);
```

```
/*
 *  files.h  --  John K. Estell  --  18 January 1996
 *
 *  Commonly-used routines for handling files and related functions.
 *
 *  version 1.0 - 18 January 1996
 *
 */


/* function prototypes */

extern FILE *open_file(/* char *filename, char *access, int failure_mode */);
extern char *change_extension(/* char *oldfilename, char *newextension */);
extern void create_filename(/* char *target, char *dir,
                               char *prefix, char *suffix */);
extern void create_directory(/* char *target, char *prefix */);
```

```
/*
 *  html.h  --  John K. Estell  --  17 January 1996
 *
 *  functions for generating html-fomatted files.
 *
 *  version 1.0 - 17 January 1996
 *
 */


/* function prototypes */

extern FILE *create_web_page(/* char *filename, char *title, char *banner */);
extern void close_web_page(/* FILE *fp */);
extern void make_html_string(/* char *target_str, char *source_str */);
```

```
/*
 *  params.h  --  John K. Estell  --  24 January 1996
 *
 *  routines for handling common actions involving the parsing
 *  of command line parameters.
 */

extern int check_num_args(/* int actual, int expected, int failure_mode */);
extern char is_flag(/* char *parameter */);
```

```
/*
 *  main.h  --  John K. Estell  --  18 January 1996
 *
 *  header file containing constants for the main file for err2html.
 *
 *  version 1.0 - 18 January 1996
 *
 */


#define GCC        1
#define LINT       2
#define GCC_EXT    "gccerr"
#define LINT_EXT   "linterr"
#define HTML_EXT   "html"
#define ERR_PREFIX "err"
#define CLEAR_FILE "dirlist"
```

```c
/*
 *  main.c  --  John K. Estell  --  21 February 1996
 *  main module for err2html utility program.
 *
 *  program for generating html-fomatted files for reporting
 *  errors and warnings for C source code files.
 *
 *  version 2.2 - 21 February 1996
 *                included output file for listing created directories;
 *                script is used to clean out err2html directories.
 *
 *  version 2.1 - 31 January 1996
 *                incorporated temporary files and directory structure.
 *
 *  version 2.0 - 18 January 1996
 *                reworked into makefile format.
 *
 *  version 1.1 - 17 January 1996
 *                Changed output so that on lines with multiple error
 *                messages all errors reported by one "analyzing tool"
 *                will be displayed before the errors found by another tool.
 *                Minor changes made to variables and added some comments.
 *
 *  version 1.0 - 10 January 1996
 *                basic setup.  Given filename of source code
 *                file, will run gcc and lint, take the resultant
 *                errors, and incorporate those into links in an
 *                html version of the source code.  The line alledged
 *                to have the error will be highlighted, and by traversing
 *                the link the error/warning messages are accessed.
 *                All operations occur in the current directory - this needs
 *                to be changed in a later version.
 *
 *
 */

#include <stdio.h>
#include "main.h"
#include "files.h"
#include "html.h"
#include "params.h"
#include "constants.h"

/* function prototypes */

int  get_line_number(int type, FILE *fp, char *string);
int  find_first_char(char *string);
void make_system_calls(char *sourcefilename);
```

```
/* main function */

int main(int argc, char *argv[])
{
   /* file pointers */

   FILE *fp_lint;              /* lint output file */
   FILE *fp_gcc;               /* gcc errors and warnings output file */
   FILE *fp_source;            /* input source code file */
   FILE *fp_html;              /* web page for program */
   FILE *fp_err;               /* web page for errors  */
   FILE *fp_clear;             /* file containing directory list */

   /* text buffers */

   char bfr_lint[BUFSIZ];
   char bfr_gcc[BUFSIZ];
   char bfr_source[BUFSIZ];
   char bfr_html[BUFSIZ];
   char bfr_err[BUFSIZ];
   char bfr_directory[BUFSIZ];  /* directory for err2html files */
   char buffer[BUFSIZ];         /* generic buffer */

   /* line pointers - keeps track of current line number in file */

   int ptr_lint = 0;
   int ptr_gcc = 0;
   int ptr_source = 0;

   /* error counter */

   int err_count = 0;

   /* generic variables */

   int i, j, page_flag;
```

```
/*** code ***/

/* handle passed parameters */

thisprogram = argv[0];
check_num_args(argc, 2, FATAL);  /* abort if we don't have exactly 2 args */

/* perform the system calls to gcc and lint */

make_system_calls(argv[1]);

/* open the appropriate files */

fp_source = open_file(argv[1], "r", FATAL);
fp_gcc    = open_file(change_extension(argv[1], GCC_EXT), "r", NON_FATAL);
fp_lint   = open_file(change_extension(argv[1], LINT_EXT), "r", NON_FATAL);
fp_html   = create_web_page(change_extension(argv[1], HTML_EXT),
                            argv[1], argv[1]);
fp_clear  = open_file(CLEAR_FILE, "a", NON_FATAL);

/* create the temporary directory and record its name... */

create_directory(bfr_directory, ERR_PREFIX);
strncpy(buffer, bfr_directory, strlen(bfr_directory) - 1); /* remove slash */
fprintf(fp_clear, "%s\n", buffer);
fclose(fp_clear);

/* set up line pointers to first reported errors */

ptr_gcc = get_line_number(GCC, fp_gcc, bfr_gcc);
ptr_lint = get_line_number(LINT, fp_lint, bfr_lint);

/* get input from the source file - modify strings as required... */

fprintf(fp_html, "<PRE>\n");

while (fgets(bfr_source, BUFSIZ, fp_source) != NULL)
{
   ptr_source++;
   make_html_string(bfr_html, bfr_source);
   page_flag = CLEAR; /* is set to 1 if an error web page is generated */

   /*
    *  if we have reached a line with a reported error, then handle all
    *  errors found on that line
    */
```

```
      if ((ptr_source == ptr_gcc) || (ptr_source == ptr_lint))
      {
         while ((ptr_source == ptr_gcc) || (ptr_source == ptr_lint))
         {
            if (page_flag == CLEAR)  /* create new error page */
            {
               err_count++;
               page_flag = SET;
               sprintf(buffer, "%s: line %d", argv[1], ptr_source);
               create_filename(bfr_err, bfr_directory, ERR_PREFIX, HTML_EXT);
               fp_err = create_web_page(bfr_err, buffer, buffer);

               /* properly indent the source code, then give linked statement */

               i = find_first_char(bfr_html);
               for (j = 0; j < i; j++)
                  fputc(' ', fp_html);   /* indentation */
               fprintf(fp_html, "<A HREF=\"%s\">%s</A>\n",
                       bfr_err, bfr_html + i);

               /* copy the statement to the error file */

               fprintf(fp_err, "<HR>\n<PRE>\n%s\n</PRE>\n<HR>\n", bfr_html + i);
            }

            /* check to see who is reporting the error */

            if (ptr_source == ptr_gcc)
            {
               make_html_string(buffer, bfr_gcc);
               fprintf(fp_err, "gcc: %s\n<P>\n", buffer);
               ptr_gcc = get_line_number(GCC, fp_gcc, bfr_gcc); /* of next error */
               continue;
            }

            if (ptr_source == ptr_lint)
            {
               make_html_string(buffer, bfr_lint);
               fprintf(fp_err, "lint: %s\n<P>\n", buffer);
               ptr_lint = get_line_number(LINT, fp_lint, bfr_lint); /* of next error */
               continue;
            }
         }
         close_web_page(fp_err);
      }
      else /* source line is free of any reported errors */
         fprintf(fp_html, "%s\n", bfr_html);
   }

   fprintf(fp_html, "</PRE>\n");
   close_web_page(fp_html);
}
```

```
/*** function definitions ***/


/*
 *  find_first_char: returns position of first non-whitespace character.
 */

int find_first_char(char *string)
{
   int i = 0;

   while (isspace(string[i]))
      i++;

   return i;
}




/*
 *  make_system_calls: invokes calls to the gcc compiler and the
 *                     lint source code checker.  Note that system()
 *                     uses /bin/sh (Bourne shell) when executing.
 */

void make_system_calls(char *sourcefilename)
{
   char buffer[BUFSIZ];

   sprintf(buffer, "gcc -Wall %s 2> %s",
                   sourcefilename, change_extension(sourcefilename, GCC_EXT));
   system(buffer);
   sprintf(buffer, "lint %s 1> %s",
                   sourcefilename, change_extension(sourcefilename, LINT_EXT));
   system(buffer);
}
```

```
/*
 *  get_line_number: reads in a line from the specified file and looks using the provided
 *                   format for a line number referencing an error/warning.  Returns line
 *                   number or -1 if nothing found.
 */

int get_line_number(int type, FILE *fp, char *string)
{
   int valid = 0;
   int i, lineno;
   char buffer[BUFSIZ];

   if (fp == NULL)
      return -1;

   strcpy(string, "error found"); /* have message just in case... */

   while (valid < 1)
   {
      if (fgets(buffer, BUFSIZ, fp) == NULL)
         break;
      switch (type)
      {
         case GCC:
                    i = 0;
                    while (buffer[i++] != ':')
                       if (buffer[i] == '\0')
                          break;
                    valid = sscanf(buffer + i, "%d", &lineno); /* read line # */
                    while (buffer[i++] != ':')
                       if (buffer[i] == '\0')
                          break;
                    strcpy(string, buffer + i); /* copy just the message */
                    break;

         case LINT:
                    i = 0;
                    while (buffer[i++] != '(')
                       if (buffer[i] == '\0')
                          break;
                    valid = sscanf(buffer + i, "%d", &lineno); /* read line # */
                    while (buffer[i++] != ':')
                       if (buffer[i] == '\0')
                          break;
                    strcpy(string, buffer + i); /* copy just the message */
                    break;
      }
   }

   if (valid < 1)
      return -1;
   else
      return lineno;
}
```

```
/*
 *  constants.c  --  John K. Estell  --  31 January 1996
 *
 *  dedicated to functions that are constantly found in my programs....
 */

#include "constants.h"

/*
 *  report_fatal_error: prints out program name and our reason for
 *                      killing off this process.
 */

void report_fatal_error(char *string)
{
   fprintf(stderr, "%s: %s\n", thisprogram, string);
   exit(1);
}
```

```
/*
 *  files.c  --  John K. Estell  --  18 January 1996
 *
 *  Commonly-used routines for handling files and functions related
 *  to files.
 *
 *  version 1.0 - 18 January 1996
 *
 */

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include "constants.h"
#include "files.h"

/* function prototypes */


/*
 *  open_file: opens specified file for specified access;
 *             incorporates error checking.  Can specify errors to
 *             be fatal or non-fatal through use of FATAL and NON_FATAL
 *             constants passed in failure_mode field.
 */

FILE *open_file(char *filename, char *access, int failure_mode)
{
   FILE *fp;

   fp = fopen(filename, access);
   if ((fp == NULL) && (failure_mode == FATAL))
      report_fatal_error("error opening file.");
   else
      return fp;
}
```

```
/*
 *  change_extension: given foo.ext, will strip off ext and replace with
 *                    new extension.  If no extension is given with foo,
 *                    will add specified new extension.
 */

char *change_extension(char *oldfilename, char *newextension)
{
   static char buffer[BUFSIZ];
   char *ptr_buf = buffer;

   strcpy(buffer, oldfilename);

   /*
    *  find either end of string or start of extension.
    *  assumes format of 'filename' or 'filename.ext'
    */

   while ((*ptr_buf != '\0') && (*ptr_buf != '.'))
      ptr_buf++;

   if (*ptr_buf == '.') /* found extension */
      ptr_buf++;
   else /* create extension */
      *ptr_buf++ = '.';

   strcpy(ptr_buf, newextension); /* append the new extension */
   return buffer;
}
```

```
/*
 *  create_filename: will attempt to create unique filenames for
 *                   temporary files.  User has option of supplying
 *                   a prefix and a suffix; routine will supply a
 *                   5-position number, which should suffice...
 *                   if the dir field specifies a path, then it must
 *                   include the final directory slash immediately
 *                   preceding the actual filename.
 */

void create_filename(char *target, char *dir, char *prefix, char *suffix)
{
   static int count = 0;  /* will keep a running count of function use */
   char buf[10];

   target[0] = '\0';
   if (strlen(dir) > 0) /* then copy the path */
      strcat(target, dir);
   if (strlen(prefix) > 0) /* then copy the prefix */
      strcat(target, prefix);
   sprintf(buf, "%05d", count++); /* output the varying portion of the name */
   strcat(target, buf);
   if (strlen(suffix) > 0) /* then copy the suffix as an extension */
   {
      strcat(target, ".");
      strcat(target, suffix);
   }
}


/*
 * create_directory: this function will create a subdirectory with a
 *                   unique name based on the pid of the current process.
 *                   The directory is created in the working directory and
 *                   its name is returned in the target array.  Access
 *                   modes hardcoded for drwxr-xr-x.  Returned string
 *                   will have '/' appended at end.
 */

void create_directory(char *target, char *prefix)
{
   int result;
   pid_t pid;

   pid = getpid();
   sprintf(target, "%s%d", prefix, pid);
   result = mkdir(target, 0755);
   if (result == -1)
   {
      fprintf(stderr, "%s: error in create temp directory\n", thisprogram);
      exit(1);
   }
   strcat(target, "/");
}
```

err2html: source code for html functions

```
 /*
 *  html.c  --  John K. Estell  --  17 January 1996
 *
 *  functions for generating html-fomatted files.
 *
 *  version 1.0 - 17 January 1996
 *
 */

#include <stdio.h>
#include "constants.h"
#include "files.h"
#include "html.h"

/*** function definitions ***/


/*
 *  make_html_string: converts ASCII characters considered special to html
 *                    into their html metacharacter equivalents.  Regular
 *                    characters are passed unchanged.
 */

void make_html_string(char *target_str, char *source_str)
{
   char *target, *source;

   target = target_str;
   source = source_str;

   do /* until end of source string */
   {
      switch (*source)
      {
         case '\n': break;
         case '<' : strncpy(target, "&lt", 3); target += 3; break;
         case '>' : strncpy(target, "&gt", 3); target += 3; break;
         case '&' : strncpy(target, "&amp", 4); target += 4; break;
         default  : *target++ = *source; break;
      }
   } while ( *source++ != '\0');
}
```

```
/*
 *  create_web_page: opens specified file and writes out initial html
 *                   formatted code for the page.
 */

FILE *create_web_page(char *filename, char *title, char *banner)
{
   FILE *fp;

   fp = open_file(filename, "w", FATAL);
   fprintf(fp, "<HEAD><TITLE>%s</TITLE></HEAD>\n", title);
   fprintf(fp, "<BODY>\n");
   fprintf(fp, "<H2>%s</H2>\n", banner);
   return fp;
}



/*
 *  close_web_page: adds final html codes and closes the file.
 */

void close_web_page(FILE *fp)
{
   fprintf(fp, "</BODY>\n");
   fclose(fp);
}
```

err2html: source code for params functions

```
/*
 *  params.c  --  John K. Estell  --  24 January 1996
 *
 *  routines for handling common actions involving the parsing
 *  of command line parameters.
 */

#include "params.h"
#include "constants.h"

/*
 *  check_num_args: compares actual number of arguments to an expected
 *                  number of arguments.  If fatal mode is used, program
 *                  will exit if the values don't agree.  In non-fatal
 *                  mode, routine will return 0 for equality, -1 for
 *                  fewer than expected arguments, and +1 for more than
 *                  expected arguments.
 */

int check_num_args(int actual, int expected, int failure_mode)
{
   if (failure_mode == FATAL)
   {
      if (actual == expected)
         return 0;
      if (actual < expected)
         report_fatal_error("too few arguments");
      if (actual > expected)
         report_fatal_error("too many arguments");
   }
   else /* non-fatal */
   {
      if (actual == expected)
         return 0;
      if (actual < expected)
         return -1;
      if (actual > expected)
         return 1;
   }
}

/*
 *  is_flag: parses passed argument string.  If it is a flag, then
 *           it will return the first character after the flag;
 *           otherwise, the NUL character is returned.
 */

char is_flag(char *parameter)
{
   if (*parameter++ == '-')
      return *parameter;
   else
      return '\0';
}
```

```
thor> cat test.c
/*
 *  test.c - John K. Estell - 8 January 1996
 *  test program with syntactical errors for html demo
 */

#include <stdio.h>

float sum(int *array, int size);

main()
{
   int result, value, numbers[10], i;
   for (i=0; i<10; i++)
   {
     scanf("%d", value);
     numbers[i] = value;
   }
   result = sum(numbers);
   printf("sum is %d\n", result);
}
```

- run the html listing and error reporting program...

```
thor> err2html test.c
```

- for background information, let's look at the generated error output from gcc and lint.

```
thor> cat test.gccerr
ld.so: warning: /usr/lib/libc.so.1.7 has older revision than expected 8
ld.so: warning: /usr/lib/libc.so.1.7 has older revision than expected 8
ld.so: warning: /usr/lib/libc.so.1.7 has older revision than expected 8
test.c:11: warning: return-type defaults to ‘int'
test.c: In function ‘main':
test.c:15: warning: implicit declaration of function ‘scanf'
test.c:15: warning: format argument is not a pointer (arg 2)
test.c:18: too few arguments to function ‘sum'
test.c:19: warning: implicit declaration of function ‘printf'
test.c:20: warning: control reaches end of non-void function
```

```
thor> cat test.linterr
test.c(8): syntax error at or near type word "int"
test.c(15): warning: value may be used before set
test.c(20): warning: main() returns random value to invocation environment
scanf returns value which is always ignored
sum used( test.c(18) ), but not defined
printf returns value which is always ignored
```

```
thor> ls
dirlist
err9468/
err9491/
err2html*
errclear*
test.c
test.gccerr
test.html
test.linterr
```

- dirlist contains a list of all temporary directories generated by err2html

```
thor> cat dirlist
err9468
err9491
```

- let's look at the contents of one of the directories

```
thor> ls err9491
err00000.html  err00002.html  err00004.html
err00001.html  err00003.html  err00005.html
```

- HTML source code file generated by err2html:

```
thor> cat test.html
<HEAD><TITLE>test.c</TITLE></HEAD>
<BODY>
<H2>test.c</H2>
<PRE>
/*
 *  test.c - John K. Estell - 8 January 1996
 *  test program with syntactical errors for html demo
 */

#include &ltstdio.h&gt

<A HREF="err9491/err00000.html">float sum(int *array, int size);</A>

main()
<A HREF="err9491/err00001.html">{</A>
   int result, value, numbers[10], i;
   for (i=0; i&lt10; i++)
   {
     <A HREF="err9491/err00002.html">scanf("%d", value);</A>
     numbers[i] = value;
   }
   <A HREF="err9491/err00003.html">result = sum(numbers);</A>
   <A HREF="err9491/err00004.html">printf("sum is %d\n", result);</A>
<A HREF="err9491/err00005.html">}</A>

</PRE>
</BODY>
```

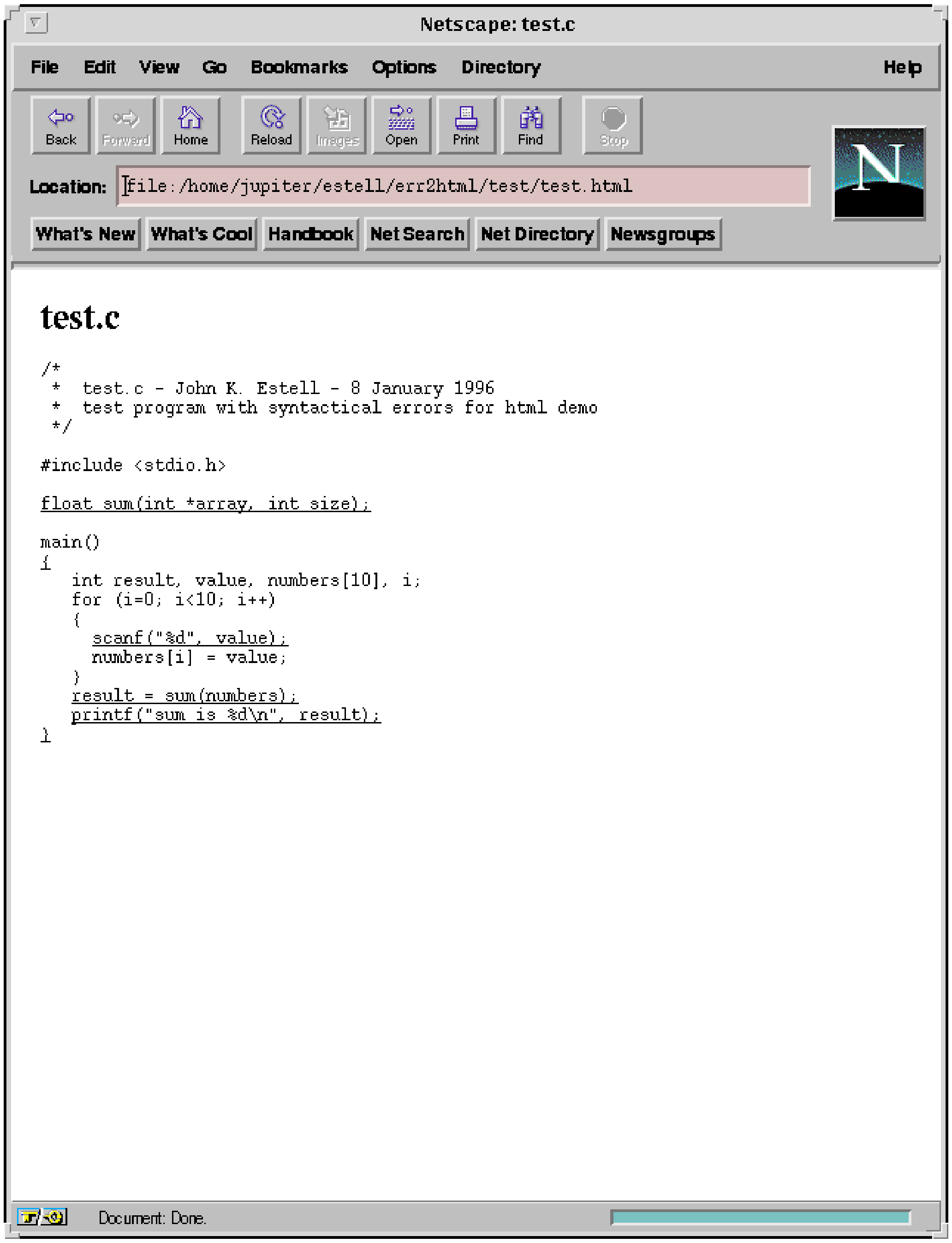
- HTML error/warning reporting files generated by err2html:

```
thor> cat err9491/err00002.html
<HEAD><TITLE>test.c: line 15</TITLE></HEAD>
<BODY>
<H2>test.c: line 15</H2>
<HR>
<PRE>
scanf("%d", value);
</PRE>
<HR>
gcc:  warning: implicit declaration of function ‘scanf'
<P>
gcc:  warning: format argument is not a pointer (arg 2)
<P>
lint:  warning: value may be used before set
<P>
</BODY>
```

```
thor> cat err9491/err00003.html
<HEAD><TITLE>test.c: line 18</TITLE></HEAD>
<BODY>
<H2>test.c: line 18</H2>
<HR>
<PRE>
result = sum(numbers);
</PRE>
<HR>
gcc:  too few arguments to function ‘sum'
<P>
</BODY>
```

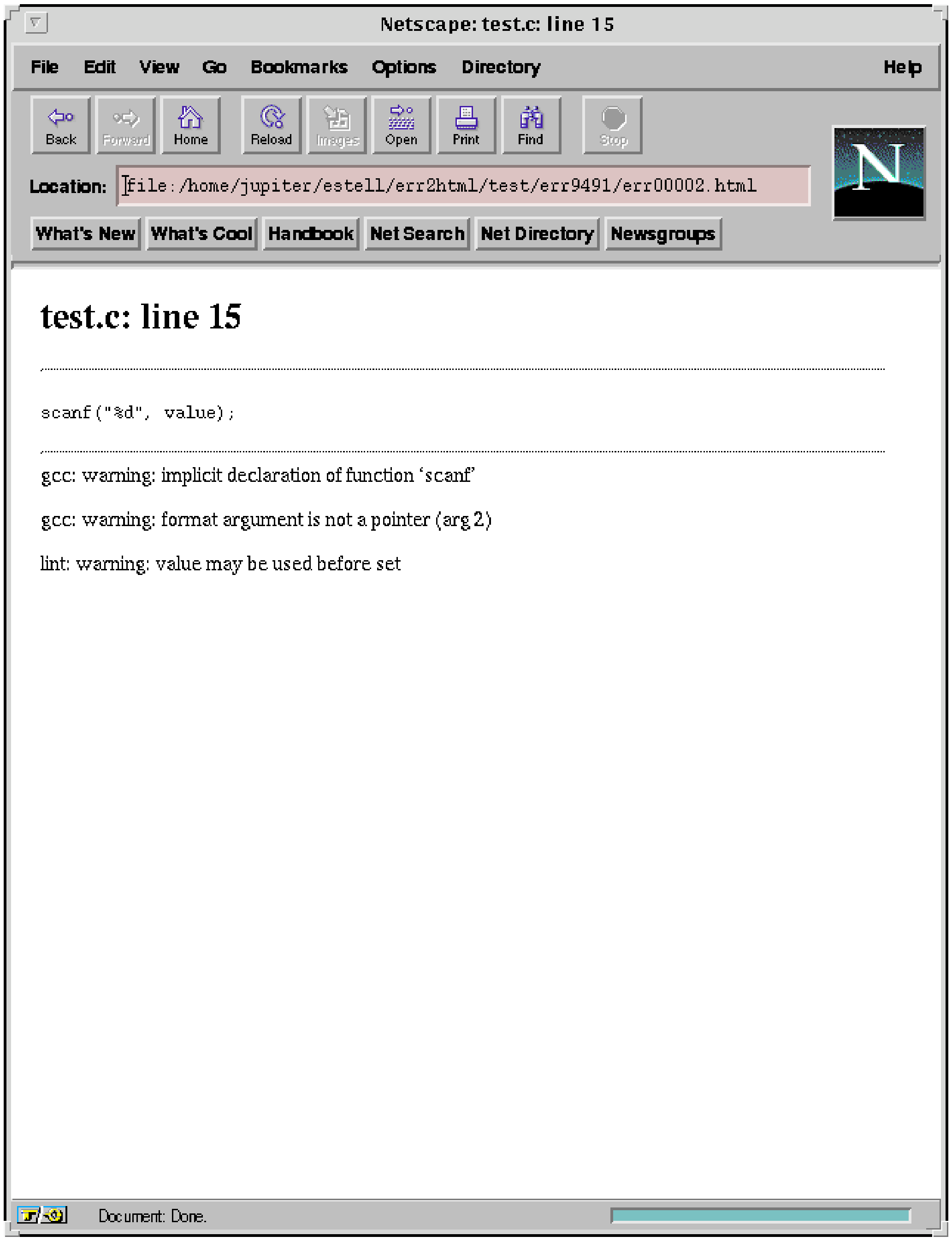
err2html: shell script for removing temporary directories

```
thor> cat errclear
#! /bin/csh -f

#
# errclear - John K. Estell - 21 February 1996
# err2html shell script routine
# used to take directory entries in clear.dirlist file
# and clean out the contents of each directory, then
# removes the directories.
#

if (-e dirlist) then
  set list = ‘cat dirlist‘

  foreach entry ($list)
    if (-d $entry) then
      cd $entry
      rm *
      cd ..
      rmdir $entry
    endif
  end

  rm dirlist
else
  echo "no directories to clean"
endif

thor> cat dirlist
err9468
err9491

thor> ls
dirlist
err9468/
err9491/
err2html*
errclear*
test.c
test.gccerr
test.html
test.linterr

thor> errclear
thor> errclear
no directories to clean
thor> ls
err2html*
errclear*
test.c
test.gccerr
test.html
test.linterr
```


Netscape: test.c
File Edit View Go Bookmarks Options Directory Help
Back Forward Home Reload Images Open Print Find Stop
Location: file:/home/jupiter/estell/err2html/test/test.html
What's New What's Cool Handbook Net Search Net Directory Newsgroups
Document: Done.

test.c

```
/*
 *  test.c - John K. Estell - 8 January 1996
 *  test program with syntactical errors for html demo
 */

#include <stdio.h>

float sum(int *array, int size);

main()
{
   int result, value, numbers[10], i;
   for (i=0; i<10; i++)
   {
     scanf("%d", value);
     numbers[i] = value;
   }
   result = sum(numbers);
   printf("sum is %d\n", result);
}
```



```
file:/home/jupiter/estell/err2html/test/err9491/err00002.html
```

test.c: line 15

```
scanf("%d", value);
```

gcc: warning: implicit declaration of function ‘scanf’

gcc: warning: format argument is not a pointer (arg 2)

lint: warning: value may be used before set

```
/*
 *  AN ELECTRONIC CHRISTMAS/CHANUKAH CARD TO ALL
 *
 *  Lyrics:         Evan Leibovitch <evan@telly.on.ca>
 *  Code:           Beverly Erlebacher <erlebach@cs.toronto.edu>
 *  Written:        December 1989
 *
 *  This code is in the public domain.
 */

#include <stdio.h>

#define DEFAULT_HOLIDAY "CHRISTMAS/CHANUKAH"
#define DAYS_OF_NEGLECT 12

char heading[] = "AN ELECTRONIC %s CARD TO ALL\n\n\
    Lyrics: Evan Leibovitch <evan@telly.on.ca>\n\
    Code: Beverly Erlebacher <erlebach@cs.toronto.edu>\n\n\
    Dedicated to Dave Mason, Chris Siebenmann, and anyone who's left\n\
    their computers alone just long enough for them to self-destruct:\n\n\
    (Sung to the tune of something or other...)\n\n";

char * cardinal[] = {"And a", "Two", "Three", "Four", "Five",
    "Six", "Seven", "Eight", "Nine", "Ten", "Eleven", "Twelve"};

char * ordinal[] = {"first", "second", "third", "fourth", "fifth", "sixth",
    "seventh", "eighth", "ninth", "tenth", "eleventh", "twelfth", "thirteenth"};

char * item[] = { "burnt-out V.D.T.", "faulty tapes;",
    "heads crashed;", "bad blocks;", "core dumps;", "bad controllers; ",
    "blown partitions;", "gettys dying;", "floppies frying;",
    "ports a-jamming;", "chips a-smoking;", "boards a-blowing;" };

char daystr[] = "\nOn the %s day I left it, my Unix gave to me:\n";

char finale[] =
   "\nOn the %s day I started adapting my Nintendo for the VME bus.\n";

main(int argc, char *argv[])
{
        int i, j;

        printf(heading, argc > 1 ? argv[1] : DEFAULT_HOLIDAY);
        for (i = 0; i < DAYS_OF_NEGLECT; i++) {
                printf(daystr, ordinal[i]);
                if (i == 0)
                        printf("\tA %s\n", item[i]);
                else for (j = i; j >= 0; j--)
                        printf("\t%s %s\n", cardinal[j], item[j]);
        }
        printf(finale, ordinal[DAYS_OF_NEGLECT]);
        exit(0);
}
```

```
thor> a.out
AN ELECTRONIC CHRISTMAS/CHANUKAH CARD TO ALL

    Lyrics: Evan Leibovitch <evan@telly.on.ca>
    Code: Beverly Erlebacher <erlebach@cs.toronto.edu>

    Dedicated to Dave Mason, Chris Siebenmann, and anyone who's left
    their computers alone just long enough for them to self-destruct:

    (Sung to the tune of something or other...)


On the first day I left it, my Unix gave to me:
        A burnt-out V.D.T.

On the second day I left it, my Unix gave to me:
        Two faulty tapes;
        And a burnt-out V.D.T.

On the third day I left it, my Unix gave to me:
        Three heads crashed;
        Two faulty tapes;
        And a burnt-out V.D.T.
```

(versus four through eleven omitted...)

```
On the twelfth day I left it, my Unix gave to me:
        Twelve boards a-blowing;
        Eleven chips a-smoking;
        Ten ports a-jamming;
        Nine floppies frying;
        Eight gettys dying;
        Seven blown partitions;
        Six bad controllers;
        Five core dumps;
        Four bad blocks;
        Three heads crashed;
        Two faulty tapes;
        And a burnt-out V.D.T.

On the thirteenth day I started adapting my Nintendo for the VME bus.
```

```
thor> cat hacker.c
#include <stdio.h>
#define O (b=b?b-1:(p++,5),*p&1<<b)
#define o O?O
char*p,j=2,b,c;e(n){for(p="|'8I0>+@{=#_P0-]PV.]F>TM!YK'?? |T\"Z8}aE<&D-!:-T'\"\
O<~cG5$,<2'#;/UI.0{d^HV6817-2F95-T7X|c^/1XB]*)3WHG0/0}dN>G RMZB.12.P] ~hM^J\\[\
<R^ (7;)R9A78{gU!:N)E5OPUR><29A6|e&9V;E[Q:,S1.P] }eES.$Z):B.*O+$G_ ~fWU8)75?I#\
75?WHN0{jE=]<V*1]JI#5VK)R9A6~J5X9X#69/+VX4 =S%!X-[)OE #1XRZ\"?~%^-#Dz&M\\RST|%\
G66*~&^HV0> {%^-8_P}%N>FO(}'M^JQ=z&U!:O(J{%&9G4|%ERO(~(WU8)G4{'E=]^G4",b=n;*p++
<122||--b;);c= *p;while(--c>31&&c!=79)putchar(44+(o?o?o?-34:68:O?60:74:O?64:o?o?
2:54:O?23:63:77:O?55:o?76:15:35:-12:o?61:O?56:65:O?66:53:o?o?O?75:58:0:70:57:o?
71:o?73:1:67:O?72:59));c>32?e(n-1):0;}main(){while(++j<15)e(1),e(13+j),e(15),e(
j-(j<4));}
thor> gcc -o hacker hacker.c
thor> hacker
On the first day of Christmas my true love gave to me
a partridge in a pear tree.

On the second day of Christmas my true love gave to me
two turtle doves
and a partridge in a pear tree.

On the third day of Christmas my true love gave to me
three french hens, two turtle doves
and a partridge in a pear tree.

On the fourth day of Christmas my true love gave to me
four calling birds, three french hens, two turtle doves
and a partridge in a pear tree.

On the fifth day of Christmas my true love gave to me
five golden rings;
four calling birds, three french hens, two turtle doves
and a partridge in a pear tree.
```

(verses six through ten omitted...)

```
On the eleventh day of Christmas my true love gave to me
eleven pipers piping, ten lords a-leaping,
nine ladies dancing, eight maids a-milking, seven swans a-swimming,
six geese a-laying, five golden rings;
four calling birds, three french hens, two turtle doves
and a partridge in a pear tree.

On the twelfth day of Christmas my true love gave to me
twelve drummers drumming, eleven pipers piping, ten lords a-leaping,
nine ladies dancing, eight maids a-milking, seven swans a-swimming,
six geese a-laying, five golden rings;
four calling birds, three french hens, two turtle doves
and a partridge in a pear tree.

thor>
```

- please don’t try to write your code like this!!!

Laurie Sue's Poetry Corner

by Laurie Sue Holstein --
A bovine with an attitude.

"Write in C"

When I find my code in tons of trouble,
Friends and colleagues come to me,
Speaking words of wisdom:
"Write in C."

As the deadline fast approaches,
And bugs are all that I can see,
Somewhere, someone whispers:
"Write in C."

Write in C, write in C,
Write in C, oh, write in C.
LISP is dead and buried,
Write in C.

I used to write a lot of FORTRAN,
For science it worked flawlessly;
Try using it for graphics!
Write in C.

If you've just spent thirty hours
Debugging Intel assembly,
Soon you will be glad to
Write in C.

Write in C, write in C,
Write in C, oh, write in C.
Nerdy wimps use BASIC,
Write in C.

You'll find that double pluses
Aren't all that they are claimed to be,
When you're done with all your classes,
Write in C.

Write in C, write in C,
Write in C, oh, write in C.
Pascal is for losers,
Write in C.

Write in C, write in C,
Write in C, oh, write in C.
Don't even mention COBOL,
Write in C.

Write in C, write in C,
Write in C, oh, write in C.
Ada goes on forever,
Write in C.

index