Chapter 2

PE 2-1

```
/* Programming Exercise 2-1  */
#include <stdio.h>

int main(void)
{
    printf("Anton Bruckner\n");
    printf("Anton\nBruckner\n");
    printf("Anton ");
    printf("Bruckner\n");

    return 0;
}
```

PE 2-3

```
/* Programming Exercise 2-3  */
#include <stdio.h>

int main(void)
{
    int ageyears;    /* age in years */
    int agedays;    /* age in days  */
                /* large ages may require the long type */
    ageyears = 44;
    agedays = 365 * ageyears;
    printf("An age of %d years is %d days.\n", ageyears, agedays);

    return 0;
}
```

PE 2-4

```
/* Programming Exercise 2-4  */
#include <stdio.h>

void jolly(void);
void deny(void);

int main(void)
{
    jolly();
    jolly();
    jolly();
    deny();
    return 0;
}

void jolly(void)
{
    printf("For he's a jolly good fellow!\n");
}

void deny(void)
{
    printf("Which nobody can deny!\n");
}
```

PE 2-5

```
/* Programming Exercise 2-5  */
#include <stdio.h>

int main(void)
{
    int toes;

    toes = 10;

    printf("toes = %d\n", toes);
    printf("Twice toes = %d\n", 2 * toes);
    printf("toes squared = %d\n", toes * toes);

    return 0;
}

/* or create two more variables, set them to 2 * toes and toes * toes */
```

PE 2-7

```
/* Programming Exercise 2-7  */
#include <stdio.h>

void one_three(void);
void two(void);
int main(void)
{
    printf("starting now:\n");
    one_three();
    printf("done!\n");
    return 0;
}

void one_three(void)
{
    printf("one\n");
    two();
    printf("three\n");
}

void two(void)
{
    printf("two\n");
}
```

Chapter 3

PE 3-2

```
/* Programming Exercise 3-2  */
#include <stdio.h>

int main(void)
{
    int ascii;

    printf("Enter an ASCII code: ");
    scanf("%d", &ascii);
    printf("%d is the ASCII code for %c.\n", ascii, ascii);

    return 0;
}
```

PE 3-4

```
/* Programming Exercise 3-4  */
#include <stdio.h>

int main(void)
{
    float num;
    printf("Enter a floating-point value: ");
    scanf("%f", &num);
    printf("fixed-point notation: %f\n", num);
    printf("exponential notation: %e\n", num);

    return 0;
}
```

PE 3-6

```
/* Programming Exercise 3-6  */
#include <stdio.h>
```

```
int main(void)
{
    float mass_mol = 3.0e-23;   /* mass of water molecule in grams */
    float mass_qt = 950;        /* mass of quart of water in grams */
    float quarts;
    float molecules;

    printf("Enter the number of quarts of water: ");
    scanf("%f", &quarts);
    molecules = quarts * mass_qt / mass_mol;
    printf("%f quarts of water contain %e molecules.\n", quarts,
            molecules);

    return 0;
}
```

Chapter 4

PE 4-1

```
/* Programming Exercise 4-1  */
#include <stdio.h>

int main(void)
{
    char fname[40];
    char lname[40];

    printf("Enter your first name: ");
    scanf("%s", fname);
    printf("Enter your last name: ");
    scanf("%s", lname);
    printf("%s, %s\n", lname, fname);

    return 0;
}
```

PE 4-4

```
/* Programming Exercise 4-4 */
#include <stdio.h>

int main(void)
{
    float height;
    char name[40];

    printf("Enter your height in inches: ");
    scanf("%f", &height);
    printf("Enter your name: ");
    scanf("%s", name);
    printf("%s, you are %.3f feet tall\n", name, height / 12.0);

    return 0;
}
```

PE 4-6

```
/* Programming Exercise 4-6 */
#include <stdio.h>
#include <float.h>

int main(void)
{
    float ot_f = 1.0 / 3.0;
    double ot_d = 1.0 / 3.0;

    printf(" float values: ");
    printf("%.4f %.12f %.16f\n", ot_f, ot_f, ot_f);
    printf("double values: ");
    printf("%.4f %.12f %.16f\n", ot_d, ot_d, ot_d);
    printf("FLT_DIG: %d\n", FLT_DIG);
    printf("DBL_DIG: %d\n", DBL_DIG);
    return 0;
}
```

Chapter 5

PE 5-1

```
/* Programming Exercise 5-1 */
#include <stdio.h>

int main(void)
{
    const int minperhour = 60;
    int minutes, hours, mins;

    printf("Enter the number of minutes to convert: ");
    scanf("%d", &minutes);
    while (minutes > 0 )
    {
        hours = minutes / minperhour;
        mins = minutes % minperhour;
        printf("%d minutes = %d hours, %d minutes\n", minutes, hours, mins);
        printf("Enter next minutes value (0 to quit): ");
        scanf("%d", &minutes);
    }
    printf("Bye\n");

    return 0;
}
```

PE 5-3

```
/* Programming Exercise 5-3 */
#include <stdio.h>

int main(void)
{
    const int daysperweek = 7;
    int days, weeks, day_rem;

    printf("Enter the number of days: ");
    scanf("%d", &days);
    weeks = days / daysperweek;
    day_rem = days % daysperweek;

    printf("%d days are %d weeks and %d days.\n", days, weeks, day_rem);

    return 0;
}
```

PE 5-5

```
/* Programming Exercise 5-5 */
#include <stdio.h>
int main(void)                /* finds sum of first n integers */
{
  int count, sum;
  int n;

  printf("Enter the upper limit: ");
  scanf("%d", &n);
  count = 0;
  sum = 0;
  while (count++ < n)
     sum = sum + count;
  printf("sum = %d\n", sum);
  return 0;
}
```

PE 5-7

```
/* Programming Exercise 5-7 */
#include <stdio.h>
void showCube(double x);
int main(void)    /* finds cube of entered number */
{
     double val;

     printf("Enter a floating-point value: ");
     scanf("%lf", &val);
     showCube(val);

    return 0;
}

void showCube(double x)
{
    printf("The cube of %e is %e.\n", x, x*x*x );
}
```

Chapter 6

PE 6-1

```
/* pe6-1.c */
/* this implementation assumes the character codes */
/* are sequential, as they are in ASCII.           */
#include <stdio.h>
#define SIZE 26
int main( void )
{
    char lcase[SIZE];
    int i;

    for (i = 0; i < SIZE; i++)
        lcase[i] = 'a' + i;
    for (i = 0; i < SIZE; i++)
        printf("%c", lcase[i]);
    printf("\n");

    return 0;
}
```

PE 6-3

```
/* pe6-3.c */
/* this implementation assumes the character codes */
/* are sequential, as they are in ASCII.           */

#include <stdio.h>

int main( void )
{
    char let = 'F';
    char start;
    char end;

    for (end = let; end >= 'A'; end--)
    {
        for (start = let; start >= end; start--)
            printf("%c", start);
        printf("\n");
    }

    return 0;
}
```

PE 6-5

```
/* pe6-5.c */

#include <stdio.h>

int main( void )
{
    int lower, upper, index;
    int square, cube;

    printf("Enter starting integer: ");
    scanf("%d", &lower);
    printf("Enter ending integer: ");
    scanf("%d", &upper);

    printf("%5s %10s %15s\n", "num", "square", "cube");
    for (index = lower; index <= upper; index++)
    {
        square = index * index;
        cube = index * square;
        printf("%5d %10d %15d\n", index, square, cube);
    }

    return 0;
}
```

PE 6-7

```
/* pe6-7.c */

#include <stdio.h>

int main( void )
{
    double n, m;
    double res;

    printf("Enter a pair of numbers: ");

    while (scanf("%lf %lf", &n, &m) == 2)
    {
        res = (n - m) / (n * m);
        printf("(%.3g - %.3g)/(%.3g*%.3g) = %.5g\n", n, m, n, m, res);
        printf("Enter next pair (non-numeric to quit): ");
    }

    return 0;
}
```

PE 6-10

```
/* pe6-10.c */

#include <stdio.h>
#define SIZE 8
int main( void )
{
    int vals[SIZE];
    int i;

    printf("Please enter %d integers.\n", SIZE);
    for (i = 0; i < SIZE; i++)
        scanf("%d", &vals[i]);
    printf("Here, in reverse order, are the values you entered:\n");
    for (i = SIZE - 1; i > 0; i--)
        printf("%d ", vals[i]);
    printf("\n");

    return 0;
}
```

PE 6-12

```
/* pe6-12.c */
/* This version starts with the 0 power */

#include <stdio.h>

#define SIZE 8
int main( void )
{
    int twopows[SIZE];
    int i;
    int value = 1;    /* 2 to the 0 */

    for (i = 0; i < SIZE; i++)
    {
        twopows[i] = value;
        value *= 2;
    }

    i = 0;
    do
    {
        printf("%d ", twopows[i]);
        i++;
    } while (i < SIZE);
    printf("\n");

    return 0;
}
```

PE 6-13

```
/* pe-13.c */
/* Programming Exercise 6-13 */
#include <stdio.h>

#define SIZE 8
int main(void)
{
    double arr[SIZE];
    double arr_cumul[SIZE];
    int i;

    printf("Enter %d numbers:\n", SIZE);

    for (i = 0; i < SIZE; i++)
    {
        printf("value #%d: ", i + 1);
        scanf("%lf", &arr[i]);
    /* or scanf("%lf", arr + i);    */
    }

    arr_cumul[0] = arr[0];      /* set first element */
    for (i = 1; i < SIZE; i++)
        arr_cumul[i] = arr_cumul[i-1] + arr[i];

    for (i = 0; i < SIZE; i++)
        printf("%8g ", arr[i]);
    printf("\n");
    for (i = 0; i < SIZE; i++)
        printf("%8g ", arr_cumul[i]);
    printf("\n");


    return 0;
}
```

PE 6-15

```
/* pe6-15.c */

#include <stdio.h>

#define RATE_SIMP 0.10
#define RATE_COMP 0.05
#define INIT_AMT 100.0
int main( void )
```

```
{
    double daphne = INIT_AMT;
    double deidre = INIT_AMT;
    int years = 0;

    while (deidre <= daphne)
    {
        daphne += RATE_SIMP * INIT_AMT;
        deidre += RATE_COMP * deidre;
        ++years;
    }
    printf("Investment values after %d years:\n", years);
    printf("Daphne: $%.2f\n", daphne);
    printf("Deidre: $%.2f\n", deidre);

    return 0;
}
```

Chapter 7

PE 7-1

```
/* Programming Exercise 7-1 */
#include <stdio.h>

int main(void)
{
     char ch;
     int sp_ct = 0;
     int nl_ct = 0;
     int other = 0;

     while ((ch = getchar()) != '#')
     {
         if (ch == ' ')
             sp_ct++;
         else if (ch == '\n')
             nl_ct++;
         else
             other++;
     }
     printf("spaces: %d, newlines: %d, others: %d\n", sp_ct, nl_ct, other);

    return 0;
}
```

PE 7-3

```
/* Programming Exercise 7-3 */
#include <stdio.h>

int main(void)
```

```
{
     int n;
     double sumeven = 0.0;
     int ct_even = 0;
     double sumodd = 0.0;
     int ct_odd = 0;

     while (scanf("%d", &n) == 1 && n != 0)
     {
         if (n % 2 == 1)
         {
             sumodd += n;
             ++ct_odd;
         }
         else
         {
             sumeven += n;
             ++ct_even;
         }
     }
     printf("Number of evens: %d", ct_even);
     if (ct_even > 0)
         printf("  average: %g", sumeven / ct_even);
     putchar('\n');

    printf("Number of odds: %d", ct_odd);
     if (ct_odd > 0)
         printf("  average: %g", sumodd / ct_odd);
     putchar('\n');
     printf("\ndone\n");

    return 0;
}
```

PE 7-5

```
/* Programming Exercise 7-5 */
#include <stdio.h>

int main(void)
{
     char ch;
     int ct1 = 0;
     int ct2 = 0;

     while ((ch = getchar()) != '#')
     {
         switch(ch)
         {
              case '.'    :  putchar('!');
                             ++ct1;
                             break;

              case '!'    :  putchar('!');
```

```
                            putchar('!');
                            ++ct2;
                            break;
                default    :    putchar(ch);
            }
        }
        printf("%d replacements of . with !\n", ct1);
        printf("%d replacements of ! with !!\n", ct2);

    return 0;
}
```

PE 7-7

```
/* Programming Exercise 7-7 */
#include <stdio.h>

#define BASEPAY     10      /* $10 per hour         */
#define BASEHRS     40      /* hours at basepay     */
#define OVERTIME    1.5     /* 1.5 time             */
#define AMT1        300     /* 1st rate tier        */
#define AMT2        150     /* 2st rate tier        */
#define RATE1       0.15    /* rate for 1st tier    */
#define RATE2       0.20    /* rate for 2nd tier    */
#define RATE3       0.25    /* rate for 3rd tier    */
int main(void)
{
    double hours;
    double gross;
    double net;
    double taxes;

    printf("Enter the number of hours worked this week: ");
    scanf("%lf", &hours);
    if (hours <= BASEHRS)
        gross = hours * BASEPAY;
    else
        gross = BASEHRS * BASEPAY + (hours - BASEHRS) * BASEPAY * OVERTIME;
    if (gross <= AMT1)
        taxes = gross * RATE1;
    else if (gross <= AMT1 + AMT2)
        taxes = AMT1 * RATE1 + (gross - AMT1) * RATE2;
    else
        taxes = AMT1 * RATE1 + AMT2 * RATE2 + (gross - AMT1 - AMT2) * RATE3;
    net = gross - taxes;
    printf("gross: $%.2f; taxes: $%.2f; net: $%.2f\n", gross, taxes, net);

    return 0;
}
```

PE 7-9

```
/* Programmming Exercise 7-9 */
#include <stdio.h>
#define NO 0
#define YES 1
int main(void)
{
   long num;                      /* value to be checked */
   long div;                      /* potential divisors  */
   long lim;                      /* limit to values     */
   int prime;

   printf("Please enter limit to values to be checked; ");
   printf("Enter q to quit.\n");
   while (scanf("%ld", &lim) == 1 && lim > 0)
   {
         for (num = 2; num <= lim; num++)
         {
          for (div = 2, prime = YES; (div * div) <= num; div++)
             if (num % div == 0)
                prime = NO;           /* number is not prime  */
          if (prime == YES)
             printf("%ld is prime.\n", num);
      }
      printf("Please enter another limit; ");
      printf("Enter q to quit.\n");
   }
   return 0;
}
```

PE 7-11

```
/* pe7-11.c */
/* Programming Exercise 7-11 */
#include <stdio.h>
#include <ctype.h>
int main(void)
{
  const double price_artichokes = 1.25;
  const double price_beets = 0.65;
  const double price_carrots = 0.89;
  const double DISCOUNT_RATE = 0.05;

  char ch;
  double lb_artichokes;
  double lb_beets;
  double lb_carrots;
  double lb_total;

  double cost_artichokes;
  double cost_beets;
  double cost_carrots;
  double cost_total;
  double final_total;
  double discount;
  double shipping;
```

```
printf("Enter a to buy artichokes, b for beets, ");
printf("c for carrots, q to quit: ");
while ((ch = getchar()) != 'q' && ch != 'Q')
{
    if (ch == '\n')
        continue;
    while (getchar() != '\n')
        continue;
     ch = tolower(ch);
    switch (ch)
    {
        case 'a' : printf("Enter pounds of artichokes: ");
                   scanf("%lf", &lb_artichokes);
                   break;
         case 'b' : printf("Enter pounds of beets: ");
                   scanf("%lf", &lb_beets);
                   break;

         case 'c' : printf("Enter pounds of carrots: ");
                   scanf("%lf", &lb_carrots);
                   break;
      default  : printf("%c is not a valid choice.\n");
  }
  printf("Enter a to buy artichokes, b for beets, ");
  printf("c for carrots, q to quit: ");
}

cost_artichokes = price_artichokes * lb_artichokes;
cost_beets = price_beets * lb_beets;
cost_carrots = price_carrots * lb_carrots;
cost_total = cost_artichokes + cost_beets + cost_carrots;
lb_total = lb_artichokes + lb_beets + lb_carrots;
if (lb_total <= 0)
    shipping = 0.0;
else if (lb_total < 5.0)
    shipping = 3.50;
else if (lb_total < 20)
    shipping = 10.0;
else
    shipping =  8.0 + 0.1 * lb_total;
if (cost_total > 100.0)
    discount = DISCOUNT_RATE * cost_total;
else
  discount = 0.0;
final_total = cost_total + shipping - discount;
printf("Your order:\n");
printf("%.2f lbs of artichokes at $%.2f per pound:$ %.2f\n",
          lb_artichokes, price_artichokes, cost_artichokes);
printf("%.2f lbs of beets at $%.2f per pound: $%.2f\n",
          lb_beets, price_beets, cost_beets);
printf("%.2f lbs of carrots at $%.2f per pound: $%.2f\n",
          lb_carrots, price_carrots, cost_carrots);
printf("Total cost of vegetables: $%.2f\n", cost_total);
if (cost_total > 100)
    printf("Volume discount: $%.2f\n", discount);
printf("Shipping: $%.2f\n", shipping);
```

```
  printf("Total charges: $%.2f\n", final_total);

  return 0;
}
```

Chapter 8

PE 8-1

```
/* Programming Exercise 8-1 */

#include <stdio.h>
int main(void)
{
    int ch;
    int ct = 0;

    while ((ch = getchar()) != EOF)
        ct++;
    printf("%d characters read\n", ct);

    return 0;
}
```

PE 8-3

```
/* Programming Exercise 8-3 */
/* Using ctype.h eliminates need to assume ASCII coding */
#include <stdio.h>
#include <ctype.h>
int main(void)
{
    int ch;
    int uct = 0;
    int lct = 0;

    while ((ch = getchar()) != EOF)
        if (isupper(ch))
            uct++;
        else if (islower(ch))
            lct++;
    printf("%d uppercase characters read\n", uct);
    printf("%d lowercase characters read\n", lct);

    return 0;
}

/*
 or you could use
 if (ch >= 'A' && ch <= 'Z')
```

```
     uct++;
 else if (ch >= 'a' && ch <= 'z')
     lct++;
*/
```

PE 8-5

```
/* Programming Exercise 8-5 */
/* binaryguess.c -- an improved number-guesser */
#include <stdio.h>
#include <ctype.h>
int main(void)
{
  int high = 100;
  int low = 1;
  int guess = (high + low) / 2;
  char response;

  printf("Pick an integer from 1 to 100. I will try to guess ");
  printf("it.\nRespond with a y if my guess is right, with");
  printf("\na h if it is high, and with an l if it is low.\n");
  printf("Uh...is your number %d?\n", guess);
  while ((response = getchar()) != 'y')       /* get response */
  {
      if (response == '\n')
          continue;
      if (response != 'h' && response != 'l')
      {
           printf("I don't understand that response. Please enter h for\n");
           printf("high, l for low, or y for correct.\n");
           continue;
       }

      if (response == 'h')
          high = guess - 1;
      else if (response == 'l')
          low = guess + 1;
      guess = (high + low) / 2;
    printf("Well, then, is it %d?\n", guess);
  }
  printf("I knew I could do it!\n");
  return 0;
}
```

PE 8-7

```
/* Programming Exercise 8-7 */
#include <stdio.h>
#include <ctype.h>

#define BASEPAY1    8.75    /* $8.75 per hour        */
#define BASEPAY2    9.33    /* $9.33 per hour        */
#define BASEPAY3    10.00   /* $10.00 per hour       */
```

```
#define BASEPAY4    11.20   /* $11.20 per hour      */
#define BASEHRS     40      /* hours at basepay     */
#define OVERTIME    1.5     /* 1.5 time             */
#define AMT1        300     /* 1st rate tier        */
#define AMT2        150     /* 2st rate tier        */
#define RATE1       0.15    /* rate for 1st tier    */
#define RATE2       0.20    /* rate for 2nd tier    */
#define RATE3       0.25    /* rate for 3rd tier    */

int getfirst(void);
void menu(void);

int main(void)
{
    double hours;
    double gross;
    double net;
    double taxes;
    double pay;
    char response;


    menu();
    while ((response = getfirst()) != 'q')
    {
        if (response == '\n')            /* skip over newlines     */
            continue;
        response = tolower(response);   /* accept A as a, etc.    */
        switch (response)
        {
            case 'a'  :  pay = BASEPAY1; break;
            case 'b'  :  pay = BASEPAY2; break;
            case 'c'  :  pay = BASEPAY3; break;
            case 'd'  :  pay = BASEPAY4; break;
            default   :  printf("Please enter a, b, c, d, or q.\n");
                         menu();
                         continue;   /* go to beginning of loop */
        }
        printf("Enter the number of hours worked this week: ");
        scanf("%lf", &hours);
        if (hours <= BASEHRS)
            gross = hours * pay;
        else
            gross = BASEHRS * pay + (hours - BASEHRS) * pay * OVERTIME;
        if (gross <= AMT1)
            taxes = gross * RATE1;
        else if (gross <= AMT1 + AMT2)
            taxes = AMT1 * RATE1 + (gross - AMT1) * RATE2;
        else
            taxes = AMT1 * RATE1 + AMT2 * RATE2 + (gross - AMT1 - AMT2) *
                    RATE3;
        net = gross - taxes;
        printf("gross: $%.2f; taxes: $%.2f; net: $%.2f\n", gross, taxes,
                net);
        menu();
    }
    printf("Done.\n");
    return 0;
}

void menu(void)
{
    printf("*********************************************************"
           "*********\n");
    printf("Enter the number corresponding to the desired pay rate"
           " or action:\n");
    printf("a)  $%4.2f/hr                b)  $%4.2f/hr\n", BASEPAY1,
            BASEPAY2);
    printf("c) $%5.2f/hr                d) $%5.2f/hr\n", BASEPAY3,
            BASEPAY4);
    printf("q) quit\n");
    printf("*********************************************************"
           "*********\n");
}

int getfirst(void)
{
    int ch;

    ch = getchar();
    while (isspace(ch))
        ch = getchar();
    while (getchar() != '\n')
        continue;
    return ch;
}
```

Chapter 9

PE 9-1

```
/* Programming Exercise 9-1 */
#include <stdio.h>

double min(double a, double b);
int main(void)
{
    double x, y;

    printf("Enter two numbers (q to quit): ");
    while (scanf("%lf %lf", &x, &y) == 2)
    {
        printf("The smaller number is %f.\n", min(x,y));
        printf("Next two values (q to quit): ");
    }
    printf("Bye!\n");

    return 0;
}
```

```
double min(double a, double b)
{
    return a < b ? a : b;

}
/* alternative implementation
double min(double a, double b)
{
    if (a < b)
        return a;
    else
        return b;
}
*/
```

PE 9-3

```
/* Programming Exercise 9-3 */
#include <stdio.h>

void chLineRow(char ch, int c, int r);
int main(void)
{
    char ch;
    int col, row;

    printf("Enter a character (# to quit): ");
    while ( (ch = getchar()) != '#')
    {
        if (ch == '\n')
            continue;
        printf("Enter number of columns and number of rows: ");
        if (scanf("%d %d", &col, &row) != 2)
            break;
        chLineRow(ch, col, row);
        printf("\nEnter next character (# to quit): ");
    }
    printf("Bye!\n");

    return 0;
}

void chLineRow(char ch, int c, int r)
{
    int col, row;

    for (row = 0; row < r ; row++)
    {
        for (col = 0; col < c; col++)
            putchar(ch);
        putchar('\n');
    }
    return;
}
```

PE 9-5

```
/* Programming Exercise 9-5 */
#include <stdio.h>

void larger_of(double *p1, double *p2);
int main(void)
{
    double x, y;

    printf("Enter two numbers (q to quit): ");
    while (scanf("%lf %lf", &x, &y) == 2)
    {
        larger_of(&x, &y);
        printf("The modified values are %f and %f.\n", x, y);
        printf("Next two values (q to quit): ");
    }
    printf("Bye!\n");

    return 0;
}

void larger_of(double *p1, double *p2)
{
    double temp = *p1 > *p2 ? *p1 : *p2;
    *p1= *p2 = temp;
}
```

PE 9-7

```
/* Programming Exercise 9-7 */
#include <stdio.h>
double power(double a, int b);  /* ANSI prototype */
int main(void)
{
  double x, xpow;
  int n;

  printf("Enter a number and the integer power");
  printf(" to which\nthe number will be raised. Enter q");
  printf(" to quit.\n");
  while (scanf("%lf%d", &x, &n) == 2)
  {
       xpow = power(x,n);       /* function call           */
       printf("%.3g to the power %d is %.5g\n", x, n, xpow);
       printf("Enter next pair of numbers or q to quit.\n");
  }
  printf("Hope you enjoyed this power trip -- bye!\n");
  return 0;
}
```

```
double power(double a, int b)   /* function definition     */
{
  double pow = 1;
  int i;

  if (b == 0)
  {
      if (a == 0)
          printf("0 to the 0 undefined; using 1 as the value\n");
      pow = 1.0;
  }
  else if (a == 0)
      pow = 0.0;
  else if (b > 0)
      for(i = 1; i <= b; i++)
       pow *= a;
  else    /* b < 0 */
      pow = 1.0 / power(a, - b);
  return pow;                  /* return the value of pow  */
}
```

PE 9-9

```
/* Programming Exercise 9-9 */
#include <stdio.h>
void to_base_n(int x, int base);
int main(void)
{
  int number;
  int b;

  printf("Enter an integer (q to quit):\n");
  while (scanf("%d", &number) == 1)
  {
       printf("Enter number base (2-10): ");
       scanf("%d", &b);
     printf("Base %d equivalent: ", b);
     to_base_n(number, b);
     putchar('\n');
     printf("Enter an integer (q to quit):\n");
  }
  return 0;
}
void to_base_n(int x, int base)   /* recursive function */
{
  int r;

  r = x % base;
  if (x >= 2)
     to_base_n(x / base, base);
  putchar('0' + r);
  return;
}
```

Chapter 10

PE 10-1

```
/* Programming Exercise 10-1 */
#include <stdio.h>
#define MONTHS 12    /* number of months in a year */
#define YRS   5      /* number of years of data    */
int main(void)
{
 /* initializing rainfall data for 1990 - 1994 */
 const float rain[YRS][MONTHS] = {
 {10.2, 8.1, 6.8, 4.2, 2.1, 1.8, 0.2, 0.3, 1.1, 2.3, 6.1, 7.4},
 {9.2, 9.8, 4.4, 3.3, 2.2, 0.8, 0.4, 0.0, 0.6, 1.7, 4.3, 5.2},
 {6.6, 5.5, 3.8, 2.8, 1.6, 0.2, 0.0, 0.0, 0.0, 1.3, 2.6, 4.2},
 {4.3, 4.3, 4.3, 3.0, 2.0, 1.0, 0.2, 0.2, 0.4, 2.4, 3.5, 6.6},
 {8.5, 8.2, 1.2, 1.6, 2.4, 0.0, 5.2, 0.9, 0.3, 0.9, 1.4, 7.2}
 };
 int year, month;
 float subtot, total;

 printf(" YEAR    RAINFALL  (inches)\n");
 for (year = 0, total = 0; year < YRS; year++)
 {             /* for each year, sum rainfall for each month */
    for (month = 0, subtot = 0; month < MONTHS; month++)
       subtot += *(*(rain + year) + month);
    printf("%5d %15.1f\n", 1990 + year, subtot);
    total += subtot;                  /* total for all years */
 }
 printf("\nThe yearly average is %.1f inches.\n\n", total/YRS);
 printf("MONTHLY AVERAGES:\n\n");
 printf(" Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct ");
 printf(" Nov  Dec\n");

 for (month = 0; month < MONTHS; month++)
 {               /* for each month, sum rainfall over years */
    for (year = 0, subtot =0; year < YRS; year++)
       subtot += *(*(rain + year) + month);
    printf("%4.1f ", subtot/YRS);
 }
 printf("\n");
 return 0;
}
```

PE 10-3

```
/* Programming Exercise 10-3 */
#include <stdio.h>
#define LEN 10
```

```
int max_arr(const int ar[], int n);
void show_arr(const int ar[], int n);

int main(void)
{
    int orig[LEN] = {1,2,3,4,12,6,7,8,9,10};
    int max;

    show_arr(orig, LEN);
    max = max_arr(orig, LEN);
    printf("%d = largest value\n", max);

    return 0;
}

int max_arr(const int ar[], int n)
{
    int i;
    int max = ar[0];
/* don't use 0 as initial max value -- fails if all array values are neg */

    for (i = 1; i < n; i++)
        if (max < ar[i])
            max = ar[i];
    return max;
}

void show_arr(const int ar[], int n)
{
    int i;

    for (i = 0; i < n; i++)
        printf("%d ", ar[i]);
    putchar('\n');
}
```

PE 10-5

```
/* Programming Exercise 10-5 */
#include <stdio.h>
#define LEN 10

float max_diff(const float ar[], int n);
void show_arr(const float ar[], int n);

int main(void)
{
    float orig[LEN] = {1.1,2,3,4,12,6,7,8,9,10};
    float max;

    show_arr(orig, LEN);
    max = max_diff(orig, LEN);
    printf("%g = maximum difference\n", max);
    return 0;
}

float max_diff(const float ar[], int n)
{
    int i;
    float max = ar[0];
    float min = ar[0];

    for (i = 1; i < n; i++)
    {
        if (max < ar[i])
            max = ar[i];
        else if (min > ar[i])
            min = ar[i];
    }
    return max - min;
}

void show_arr(const float ar[], int n)
{
    int i;

    for (i = 0; i < n; i++)
        printf("%g ", ar[i]);
    putchar('\n');
}
```

PE 10-7

```
/* Programming Exercise 10-7 */
#include <stdio.h>
#define LEN1 7
#define LEN2 3

void copy_arr(int ar1[], const int ar2[], int n);
void show_arr(const int ar[], int n);

int main(void)
{
    int orig[LEN1] = {1,2,3,4,5,6,7};
    int copy[LEN2];

    show_arr(orig, LEN1);
    copy_arr(copy, orig + 2, LEN2);
    show_arr(copy, LEN2);

    return 0;
}

void copy_arr(int ar1[], const int ar2[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        ar1[i] = ar2[i];
}

void show_arr(const int ar[], int n)
{
    int i;

    for (i = 0; i < n; i++)
        printf("%d ", ar[i]);
    putchar('\n');
}
```

PE 10-10

```
/* Programming Exercise 10-10 */
#include <stdio.h>
#define ROWS 3
#define COLS 5

void times2(int ar[][COLS], int r);
void showarr2(int ar[][COLS], int r);

int main(void)
{
    int stuff[ROWS][COLS] = {    {1,2,3,4,5},
                                {6,7,8,9,10},
                                {11,12,13,14,15}
                            };
    showarr2(stuff, ROWS);
    putchar('\n');
    times2(stuff, ROWS);
    showarr2(stuff, ROWS);

    return 0;
}

void times2(int ar[][COLS], int r)
{
    int row, col;

    for (row = 0; row < r; row++)
        for (col = 0; col < COLS; col++)
            ar[row][col] *= 2;

}

void showarr2(int ar[][COLS], int r)
{
    int row, col;

    for (row = 0; row < r; row++)
    {
        for (col = 0; col < COLS; col++)
```

```
            printf("%d ", ar[row][col]);
        putchar('\n');
    }
}
```

PE 10-13

```
/* Programming Exercise 10-13 */
#include <stdio.h>
#define ROWS 3
#define COLS 5

void store(double ar[], int n);
double average2d(int rows, int cols, double ar[rows][cols]);
double max2d(int rows, int cols, double ar[rows][cols]);
void showarr2(int rows, int cols, double ar[rows][cols]);
double average(const double ar[], int n);

int main(void)
{
    double stuff[ROWS][COLS];
    int row;

    for (row = 0; row < ROWS; row++)
    {
        printf("Enter %d numbers for row %d\n", COLS, row + 1);
        store(stuff[row], COLS);
    }

    printf("array contents:\n");
    showarr2(ROWS, COLS, stuff);

    for (row = 0; row < ROWS; row++)
        printf("average value of row %d = %g\n", row + 1, average(stuff[row],
COLS));
    printf("average value of all rows = %g\n", average2d(ROWS, COLS, stuff));
    printf("largest value = %g\n", max2d(ROWS, COLS, stuff));
    printf("Bye!\n");
    return 0;
}

void store(double ar[], int n)
{
    int i;
    for (i = 0; i < n; i++)
    {
        printf("Enter value #%d: ", i + 1);
        scanf("%lf", & ar[i]);
    }
}


double average2d(int rows, int cols, double ar[rows][cols])
{
    int r, c;
    double sum = 0.0;

    for (r = 0; r < rows; r++)
        for (c = 0; c < cols; c++)
            sum += ar[r][c];
    if (rows * cols > 0)
        return sum / (rows * cols);
    else
        return 0.0;
}

double max2d(int rows, int cols, double ar[rows][cols])
{
    int r, c;
    double max = ar[0][0];

    for (r = 0; r < rows; r++)
        for (c = 0; c < cols; c++)
            if (max < ar[r][c])
                max = ar[r][c];
    return max;

}

void showarr2(int rows, int cols, double ar[rows][cols])
{
    int row, col;

    for (row = 0; row < rows; row++)
    {
        for (col = 0; col < cols; col++)
            printf("%g ", ar[row][col]);
        putchar('\n');
    }
}

double average(const double ar[], int n)
{
    int i;
    double sum = 0.0;

    for (i = 0; i < n; i++)
        sum += ar[i];
    if (n > 0)
        return sum / n;
    else
        return 0.0;
}
```

Chapter 11

PE 11-1

```
/* Programming Exercise 11-1 */
#include <stdio.h>
#define LEN 10
char * getnchar(char * str, int n);
int main(void)
{
    char input[LEN];
    char *chk;

    chk = getnchar(input, LEN - 1);
    if (chk == NULL)
        puts("Input failed.");
    else
        puts(input);
    puts("Done.\n");

    return 0;
}

char * getnchar(char * str, int n)
{
    int i;
    int ch;

    for (i = 0; i < n; i++)
    {
        ch = getchar();
        if (ch != EOF)
            str[i] = ch;
        else
            break;
    }
    if (ch == EOF)
        return NULL;
    else
    {
        str[i] = '\0';
        return str;
    }
}
```

PE 11-3

```
/* Programming Exercise 11-3 */
#include <stdio.h>
#define LEN 80
char * getword(char * str);
int main(void)
{
    char input[LEN];
    char *chk;
    while (getword(input) != NULL)
        puts(input);
    puts("Done.\n");

    return 0;
}

#include <ctype.h>
char * getword(char * str)
{
    int i;
    int ch;

    while ((ch = getchar()) != EOF && !isspace(ch))
        *str++ = ch;
    *str = '\0';
    if (ch == EOF)
        return NULL;
    else
    {
        while (ch != '\n')
            ch = getchar();
        return str;
    }
}
```

PE 11-5

```
/* Programming Exercise 11-5 */
#include <stdio.h>
#define LEN 80
int is_within(const char * str, char c);
int main(void)
{
    char input[LEN];
    char ch;
    int found;;

    printf("Enter a string: ");
    while (gets(input) && input[0] != '\0')
    {
        printf("Enter a character: ");
        ch = getchar();
        while (getchar() != '\n')
            continue;
        found = is_within(input, ch);
        if (found == 0)
            printf("%c not found in string.\n", ch);
        else
            printf("%c found in string %s\n", ch, input);
        printf("Next string: ");
    }
    puts("Done.\n");

    return 0;
}

int is_within(const char * str, char ch)
{
    while (*str != ch && *str != '\0')
        str++;
    return *str;   /* = 0 if \0 reached, non-zero otherwise */
}
```

PE 11-7

```
/* Programming Exercise 11-7 */
#include <stdio.h>
#define LEN 20
char * string_in(const char * s1, const char * s2);
int main(void)
{
    char orig[LEN] = "transportation";
    char * find;

    puts(orig);
    find = string_in(orig, "port");
    if (find)
        puts(find);
    else
        puts("Not found");
    find = string_in(orig, "part");
    if (find)
        puts(find);
    else
        puts("Not found");

    return 0;
}

#include <string.h>
char * string_in(const char * s1, const char * s2)
{
    int l2 = strlen(s2);
    int tries;            /* maximum number of comparisons    */
    int nomatch = 1;    /* set to 0 if match is found        */

    tries = strlen(s1) + 1 - l2;
    if (tries > 0)
        while (( nomatch = strncmp(s1, s2, l2)) && tries--)
            s1++;
    if (nomatch)
        return NULL;
    else
        return (char *) s1;  /* cast const away */
}
```

PE 11-9

```
/* Programming Exercise 11-9 */
#include <stdio.h>
#define LEN 81
int drop_space(char * s);
int main(void)
{
    char orig[LEN];

    while (gets(orig) && orig[0] != '\0')
    {
        drop_space(orig);
        puts(orig);
    }
    puts("Bye!");
    return 0;
}

int drop_space(char * s)
{
    int ct = 0;
    char * pos;
    while (*s)     /* or while (*s != '\0') */
    {
        if (*s == ' ')
        {
            pos = s;
            do
            {
                *pos = *(pos + 1);
                pos++;
            } while (*pos);
        }
        else
            s++;
    }

}
```

PE 11-11

```
/* pe11-11.c -- counts words and certain characters */
/* Programming Exercise 11-11                       */
#include <stdio.h>
#include <ctype.h>       // for isspace()
#include <stdbool.h>     // for bool, true, false
int main(void)
{
   char c;               // read in character
   int low_ct = 0;       // number of lowercase characters
   int up_ct = 0;        // number of uppercase characters
   int dig_ct = 0;       // number of digits
   int n_words = 0;      // number of words
   int punc_ct = 0;      // number of punctuation marks
   bool inword = false;  // == true if c is in a word

   printf("Enter text to be analyzed (EOF to terminate):\n");
   while ((c = getchar()) != EOF)
   {
        if (islower(c))
           low_ct++;
        else if (isupper(c))
           up_ct++;
        else if (isdigit(c))
           dig_ct++;
        else if (ispunct(c))
           punc_ct++;
      if (!isspace(c) && !inword)
      {
         inword = true;  // starting a new word
         n_words++;      // count word
      }
      if (isspace(c) && inword)
         inword = false; // reached end of word
   }
   printf("\nwords = %d, lowercase = %d, uppercase = %d, "
          "digits = %d, punctuation = %d\n",
           n_words,low_ct,up_ct, dig_ct, punc_ct);
   return 0;
}
```

PE 11-13

```
/* Programming Exercise 11-13 */
#include <stdio.h>
#include <stdlib.h>        /* for atof()           */
#include <math.h>        /* for pow()            */
/* #include <console.h> */   /* Macintosh adjustment */
int main(int argc, char *argv[])
{
    double num, exp;

    /* argc = ccommand(&argv); */  /* Macintosh adjustment */
    if (argc != 3)
        printf("Usage: %s number exponent\n", argv[0]);
    else
    {
        num = atof(argv[1]);
        exp = atof(argv[2]);
        printf("%f to the %f power = %g\n", num, exp, pow(num,exp));
    }

    return 0;
}
```

PE 11-15

```
/* Programming Exercise 11-15 */
#include <stdio.h>
#include <ctype.h>
/* #include <console.h> */   /* Macintosh adjustment */

int main(int argc, char *argv[])
{
    char mode = 'p';
    int ok = 1;
    int ch;

    /*argc = ccommand(&argv); */  /* Macintosh adjustment */

    if (argc > 2)
    {
        printf("Usage: %s [-p | -u | -l]\n", argv[0]);
        ok = 0;                 /* skip processing input */
    }
    else if (argc == 2)
    {
        if (argv[1][0] != '-')
        {
            printf("Usage: %s [-p | -u | -l]\n", argv[0]);
            ok = 0;
        }
        else
            switch(argv[1][1])
            {
                case 'p' :
                case 'u' :
                case 'l' : mode = argv[1][1];
                           break;
                default  : printf("%s is an invalid flag; ", argv[1]);
                           printf("using default flag (-p).\n");
            }
    }

    if (ok)
        while ((ch = getchar() ) != EOF)
        {
            switch(mode)
            {
                case 'p'  :  putchar(ch);
                             break;
                case 'u'  :  putchar(toupper(ch));
                             break;
                case 'l'  :  putchar(tolower(ch));
            }
        }

    return 0;
}
```

Chapter 12

PE 12-1

```
/* pe12-1.c  -- deglobalizing global.c */
/* Programming Exercise 12-1           */
/* one of several approaches */
#include <stdio.h>
void critic(int * u);
int main(void)
{
   int units;   /* units now local */

   printf("How many pounds to a firkin of butter?\n");
   scanf("%d", &units);
   while ( units != 56)
       critic(&units);
   printf("You must have looked it up!\n");
   return 0;
}

void critic(int * u)
{
   printf("No luck, chummy. Try again.\n");
   scanf("%d", u);
}

// or use a return value:
// units = critic();

// and have critic look like this:
/*
int critic(void)
{
   int u;
   printf("No luck, chummy. Try again.\n");
   scanf("%d", &u);
   return u;
}
*/

// or have main() collect the next value for units
```

PE 12-3

```
//pe12-3a.h

#define METRIC 0
#define US 1
#define USE_RECENT 2

void check_mode(int *pm);
void get_info(int mode, double * pd, double * pf);

void show_info(int mode, double distance, double fuel);

// pe12-3a.c
#include <stdio.h>
#include "pe12-3a.h"

void check_mode(int *pm)
{
    if (*pm != METRIC && *pm != US)
    {
        printf("Invalid mode specified. Mode %d\n", *pm);
        printf("Previous mode will be used.\n");
        *pm = USE_RECENT;
    }
}

void get_info(int mode, double * pd, double * pf)
{
    if (mode == METRIC)
        printf("Enter distance traveled in kilometers: ");
    else
        printf("Enter distance traveled in miles: ");
    scanf("%lf",pd);
    if (mode == METRIC)
        printf("Enter fuel consumed in liters: ");
    else
        printf("Enter fuel consumed  in gallons: ");
    scanf("%lf", pf);
}

void show_info(int mode, double distance, double fuel)
{
    printf("Fuel consumption is ");
    if (mode == METRIC)
        printf("%.2f liters per 100 km.\n", 100 * fuel / distance);
    else
        printf("%.1f miles per gallon.\n", distance / fuel);
}

// pe12-3.c
#include <stdio.h>
#include "pe12-3a.h"
int main(void)
{
  int mode;
  int prev_mode = METRIC;
  double distance, fuel;

  printf("Enter 0 for metric mode, 1 for US mode: ");
  scanf("%d", &mode);
  while (mode >= 0)
  {
        check_mode(&mode);
        if (mode == USE_RECENT)
```

```
            mode = prev_mode;
        prev_mode = mode;
      get_info(mode, &distance, &fuel);
      show_info(mode, distance, fuel);
        printf("Enter 0 for metric mode, 1 for US mode");
        printf(" (-1 to quit): ");
      scanf("%d", &mode);

  }
  printf("Done.\n");

  return 0;
}
```

PE 12-5

```
/* pe12-5.c  */
#include <stdio.h>
#include <stdlib.h>
void print(const int array[], int limit);
void sort(int array[], int limit);

#define SIZE 100
int main(void)
{
    int i;
    int arr[SIZE];

    for (i = 0; i < SIZE; i++)
        arr[i] = rand() % 10 + 1;
    puts("initial array");
    print(arr,SIZE);
    sort(arr,SIZE);
    puts("\nsorted array");
    print(arr,SIZE);

    return 0;
}

/* sort.c -- sorts an integer array in decreasing order */
void sort(int array[], int limit)
{
   int top, search, temp;

   for (top = 0; top < limit -1; top++)
       for (search = top + 1; search < limit; search++)
            if (array[search] > array[top])
            {
                 temp = array[search];
                 array[search] = array[top];
                 array[top] = temp;
            }
}

/* print.c -- prints an array */
```

```
void print(const int array[], int limit)
{
   int index;

   for (index = 0; index < limit; index++)
   {
      printf("%2d ", array[index]);
      if (index % 10 == 9)
         putchar('\n');
   }
   if (index % 10 != 0)
      putchar('\n');
}
```

PE 12-7

```
/* pe12-7.c  */
#include <stdio.h>
#include <stdlib.h>  /* for srand() */
#include <time.h>    /* for time()  */
int rollem(int);

int main(void)
{
    int dice, count, roll;
    int sides;
    int set, sets;

    srand((unsigned int) time(0));  /* randomize rand() */

    printf("Enter the number of sets; enter q to stop.\n");
    while ( scanf("%d", &sets) == 1)
    {
          printf("How many sides and how many dice?\n");
        scanf("%d %d", &sides, &dice);
        printf("Here are %d sets of %d %d-sided throws.\n", sets, dice,
                sides);
        for (set = 0; set < sets; set++)
        {
            for (roll = 0, count = 0; count < dice; count++)
                roll += rollem(sides);
                /* running total of dice pips */
            printf("%4d ", roll);
            if (set % 15 == 14)
                putchar('\n');
        }
        if (set % 15 != 0)
            putchar('\n');
        printf("How many sets? Enter q to stop.\n");
    }
    printf("GOOD FORTUNE TO YOU!\n");
    return 0;
}

int rollem(int sides)
```

```
{
    int roll;

    roll = rand() % sides + 1;
    return roll;
}
```

Chapter 13

PE 13-2

```
/* Programming Exercise 13-2 */
#include <stdio.h>
#include <stdlib.h>
//#include <console.h>    /* Macintosh adjustment */


int main(int argc, char *argv[])
{
    int byte;
    FILE * source;
    FILE * target;

//    argc = ccommand(&argv);   /* Macintosh adjustment */

    if (argc != 3)
    {
        printf("Usage: %s sourcefile targetfile\n", argv[0]);
        exit(EXIT_FAILURE);
    }


    if ((source = fopen(argv[1], "rb")) == NULL)
    {
        printf("Could not open file %s for input\n", argv[1]);
        exit(EXIT_FAILURE);
    }
    if ((target = fopen(argv[2], "wb")) == NULL)
    {
        printf("Could not open file %s for output\n", argv[2]);
        exit(EXIT_FAILURE);
    }
    while ((byte = getc(source)) != EOF)
    {
        putc(byte, target);
    }
    if (fclose(source) != 0)
        printf("Could not close file %s\n", argv[1]);
    if (fclose(target) != 0)
        printf("Could not close file %s\n", argv[2]);

    return 0;
}
```

PE 13-4

```
/* Programming Exercise 13-4 */
#include <stdio.h>
#include <stdlib.h>
#include <console.h>    /* Macintosh adjustment */


int main(int argc, char *argv[])
{
    int byte;
    FILE * source;
    int filect;

    argc = ccommand(&argv);   /* Macintosh adjustment */

    if (argc == 1)
    {
        printf("Usage: %s filename[s]\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    for (filect = 1; filect < argc; filect++)
    {
        if ((source = fopen(argv[filect], "r")) == NULL)
        {
            printf("Could not open file %s for input\n", argv[filect]);
            continue;
        }
        while ((byte = getc(source)) != EOF)
        {
            putchar(byte);
        }
        if (fclose(source) != 0)
            printf("Could not close file %s\n", argv[1]);
    }

    return 0;
}
```

PE 13-5

```
/* Programming Exercise 13-5 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
//#include <console.h>    /* Macintosh adjustment */

#define BUFSIZE 1024
#define SLEN 81
void append(FILE *source, FILE *dest);
```

```
int main(int argc, char *argv[])
{
    FILE *fa, *fs;
    int files = 0;
    int fct;

 //   argc = ccommand(&argv);   /* Macintosh adjustment */

    if (argc < 3)
    {
        printf("Usage: %s appendfile sourcefile[s]\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    if ((fa = fopen(argv[1], "a")) == NULL)
    {
        fprintf(stderr, "Can't open %s\n", argv[1]);
        exit(EXIT_FAILURE);
    }
    if (setvbuf(fa, NULL, _IOFBF, BUFSIZE) != 0)
    {
        fputs("Can't create output buffer\n", stderr);
        exit(EXIT_FAILURE);
    }

    for (fct = 2; fct < argc; fct++)
    {
        if (strcmp(argv[fct], argv[1]) == 0)
            fputs("Can't append file to itself\n",stderr);
        else if ((fs = fopen(argv[fct], "r")) == NULL)
            fprintf(stderr, "Can't open %s\n", argv[fct]);
        else
        {
            if (setvbuf(fs, NULL, _IOFBF, BUFSIZE) != 0)
            {
                fputs("Can't create output buffer\n",stderr);
                continue;
            }
            append(fs, fa);
            if (ferror(fs) != 0)
                fprintf(stderr,"Error in reading file %s.\n",
                        argv[fct]);
            if (ferror(fa) != 0)
                fprintf(stderr,"Error in writing file %s.\n",
                        argv[1]);
            fclose(fs);
            files++;
            printf("File %s appended.\n", argv[fct]);
        }
    }
    printf("Done. %d files appended.\n", files);
    fclose(fa);

    return 0;
}
```

```
void append(FILE *source, FILE *dest)
{
    size_t bytes;
    static char temp[BUFSIZE]; // allocate once

    while ((bytes = fread(temp,sizeof(char),BUFSIZE,source)) > 0)
        fwrite(temp, sizeof (char), bytes, dest);
}
```

PE 13-7

```
/* Programming Exercise 13-7a */
/* code assumes that end-of-line immediately precedes end-of-file */

#include <stdio.h>
#include <stdlib.h>
#include <console.h>    /* Macintosh adjustment */

int main(int argc, char *argv[])
{
    int ch1, ch2;
    FILE * f1;
    FILE * f2;

    argc = ccommand(&argv);   /* Macintosh adjustment */

    if (argc != 3)
    {
        printf("Usage: %s file1 file2\n", argv[0]);
        exit(EXIT_FAILURE);
    }
    if ((f1 = fopen(argv[1], "r")) == NULL)
    {
        printf("Could not open file %s for input\n", argv[1]);
        exit(EXIT_FAILURE);
    }
    if ((f2 = fopen(argv[2], "r")) == NULL)
    {
        printf("Could not open file %s for input\n", argv[2]);
        exit(EXIT_FAILURE);
    }
    ch1 = getc(f1);
    ch2 = getc(f2);

    while (ch1 != EOF || ch2 != EOF)
    {
        while (ch1 != EOF && ch1 != '\n') /* skipped after EOF reached */
        {
            putchar(ch1);
            ch1 = getc(f1);
        }
        if (ch1 != EOF)
        {
            putchar('\n');
            ch1 = getc(f1);
        }
        while (ch2 != EOF && ch2 != '\n') /* skipped after EOF reached */
        {
            putchar(ch2);
            ch2 = getc(f2);
        }

        if (ch2 != EOF)
        {
            putchar('\n');
            ch2 = getc(f2);
        }
    }

    if (fclose(f1) != 0)
        printf("Could not close file %s\n", argv[1]);
    if (fclose(f2) != 0)
        printf("Could not close file %s\n", argv[2]);

    return 0;
}

/* Programming Exercise 13-7b */
/* code assumes that end-of-line immediately precedes end-of-file */

#include <stdio.h>
#include <stdlib.h>
#include <console.h>    /* Macintosh adjustment */

int main(int argc, char *argv[])
{
    int ch1, ch2;
    FILE * f1;
    FILE * f2;

    argc = ccommand(&argv);   /* Macintosh adjustment */

    if (argc != 3)
    {
        printf("Usage: %s file1 file2\n", argv[0]);
        exit(EXIT_FAILURE);
    }
    if ((f1 = fopen(argv[1], "r")) == NULL)
    {
        printf("Could not open file %s for input\n", argv[1]);
        exit(EXIT_FAILURE);
    }
    if ((f2 = fopen(argv[2], "r")) == NULL)
    {
        printf("Could not open file %s for input\n", argv[2]);
        exit(EXIT_FAILURE);
    }
    ch1 = getc(f1);
    ch2 = getc(f2);

    while (ch1 != EOF || ch2 != EOF)
    {
        while (ch1 != EOF && ch1 != '\n') /* skipped after EOF reached */
        {
            putchar(ch1);
            ch1 = getc(f1);
        }
        if (ch1 != EOF)
        {
            if (ch2 == EOF)
                putchar('\n');
            else
                putchar(' ');
            ch1 = getc(f1);
        }
        while (ch2 != EOF && ch2 != '\n') /* skipped after EOF reached */
        {
            putchar(ch2);
            ch2 = getc(f2);
        }

        if (ch2 != EOF)
        {
            putchar('\n');
            ch2 = getc(f2);
        }
    }

    if (fclose(f1) != 0)
        printf("Could not close file %s\n", argv[1]);
    if (fclose(f2) != 0)
        printf("Could not close file %s\n", argv[2]);

    return 0;
}
```

PE 13-9

```
/* Programming Exercise 13-9 */
/* to simplify accounting, store one number and word per line */

#include <stdio.h>
#include <stdlib.h>
#define MAX 40

int main(void)
{
     FILE *fp;
     char words[MAX];
     int wordct = 0;

     if ((fp = fopen("wordy", "a+")) == NULL)
     {
          fprintf(stderr,"Can't open \"words\" file.\n");
          exit(1);
     }
     /* determine current number of entries */
     rewind(fp);
     while (fgets(words, MAX - 1, fp) != NULL)
         wordct++;
     rewind(fp);

     puts("Enter words to add to the file. Enter one word per line, and ");
     puts("press the Enter key at the beginning of a line to terminate.");
     while (gets(words) != NULL  && words[0] != '\0')
          fprintf(fp, "%d: %s\n", ++wordct, words);
     puts("File contents:");
     rewind(fp);          /* go back to beginning of file */
     while (fgets(words, MAX - 1, fp) != NULL)
          fputs(words, stdout);
     if (fclose(fp) != 0)
          fprintf(stderr,"Error closing file\n");

     return 0;
}
```

PE 13-11

```
/* Programming Exercise 13-11 */

#include <stdio.h>
#include <stdlib.h>
#include <console.h>    /* Macintosh adjustment */

#define SLEN 256
const char *errmesg[] = {"Usage: %s string filename]\n",
                         "Can't open file %s\n" };

int main(int argc, char *argv[])
{
    FILE *fp;
    char line[SLEN];

    argc = ccommand(&argv);   /* Macintosh adjustment */
    if (argc != 3)
    {
        fprintf(stderr, errmesg[0], argv[0]);
        exit(EXIT_FAILURE);
    }

    if ((fp = fopen(argv[2], "r")) == NULL)
    {
        fprintf(stderr, errmesg[1], argv[2]);
        exit(EXIT_FAILURE);
    }

    while (fgets(line, SLEN - 1, fp) != NULL)
    {
        if (strstr(line, argv[1]) != NULL)
            fputs(line, stdout);
    }
    fclose(fp);

    return 0;
}
```

PE 13-12

```
 Data for program:

0 0 9 0 0 0 0 0 0 0 0 0 5 8 9 9 8 5 2 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 9 0 0 0 0 0 0 0 5 8 9 9 8 5 5 2 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 5 8 1 9 8 5 4 5 2 0 0 0 0 0 0 0 0 0
0 0 0 0 9 0 0 0 0 0 0 0 5 8 9 9 8 5 0 4 5 2 0 0 0 0 0 0 0 0
0 0 9 0 0 0 0 0 0 0 0 0 5 8 9 9 8 5 0 0 4 5 2 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 5 8 9 1 8 5 0 0 0 4 5 2 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 5 8 9 9 8 5 0 0 0 0 4 5 2 0 0 0 0 0
5 5 5 5 5 5 5 5 5 5 5 5 5 8 9 9 8 5 5 5 5 5 5 5 5 5 5 5 5 5
8 8 8 8 8 8 8 8 8 8 8 8 5 8 9 9 8 5 8 8 8 8 8 8 8 8 8 8 8 8
9 9 9 9 0 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 3 9 9 9 9 9 9 9
8 8 8 8 8 8 8 8 8 8 8 8 5 8 9 9 8 5 8 8 8 8 8 8 8 8 8 8 8 8
5 5 5 5 5 5 5 5 5 5 5 5 5 8 9 9 8 5 5 5 5 5 5 5 5 5 5 5 5 5
0 0 0 0 0 0 0 0 0 0 0 0 5 8 9 9 8 5 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 5 8 9 9 8 5 0 0 0 0 6 6 0 0 0 0 0 0
0 0 0 0 2 2 0 0 0 0 0 0 5 8 9 9 8 5 0 0 5 6 0 0 6 5 0 0 0 0
0 0 0 0 3 3 0 0 0 0 0 0 5 8 9 9 8 5 0 5 6 1 1 1 1 6 5 0 0 0
0 0 0 0 4 4 0 0 0 0 0 0 5 8 9 9 8 5 0 0 5 6 0 0 6 5 0 0 0 0
0 0 0 0 5 5 0 0 0 0 0 0 5 8 9 9 8 5 0 0 0 0 6 6 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 5 8 9 9 8 5 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 5 8 9 9 8 5 0 0 0 0 0 0 0 0 0 0 0 0

/* Programming Exercise 13-12 */
#include <stdio.h>
#include <stdlib.h>

#define ROWS    20
#define COLS    30
#define LEVELS    10
const char trans[LEVELS + 1] = " .':~*=&%@";

void MakePic(int data[][COLS], char pic[][COLS], int rows);
void init(char arr[][COLS], char ch);

int main()
{
    int row, col;
    int picIn[ROWS][COLS];
    char picOut[ROWS][COLS];
    char fileName[40];
    FILE * infile;

    init(picOut, 'S');

    printf("Enter name of file: ");
    scanf("%s", fileName);
    if ((infile = fopen(fileName, "r")) == NULL)
    {
        fprintf(stderr, "Could not open data file.\n");
        exit(EXIT_FAILURE);
    }

    for (row = 0; row < ROWS; row++)
        for (col = 0; col < COLS; col++)
            fscanf(infile, "%d",  &picIn[row][col]);
    if (ferror(infile))
    {
        fprintf(stderr, "Error getting data from file.\n");
        exit(EXIT_FAILURE);
    }
    MakePic(picIn, picOut, ROWS);

    for (row = 0; row < ROWS; row++)
    {
        for (col = 0; col < COLS; col++)
            putchar(picOut[row][col]);
        putchar('\n');
    }
    return 0;
}

void init(char arr[][COLS], char ch)
{
    int r, c;
    for (r = 0; r < ROWS; r++)
        for (c = 0; c < COLS; c++)
            arr[r][c] = ch;
}

void MakePic(int data[][COLS], char pic[][COLS], int rows)
{
    int row, col;
    for (row = 0; row < rows; row++)
        for (col = 0; col < COLS; col++)
            pic[row][col] = trans[data[row][col]];
}
```

Chapter 14

PE 14-1

```
/* pe14-1.c  */
#include <stdio.h>
#include <string.h>
#include <ctype.h>

struct month {
    char name[10];
    char abbrev[4];
    int days;
    int monumb;
};

const struct month months[12] = {
    {"January", "Jan", 31, 1},
    {"February", "Feb", 28, 2},
    {"March", "Mar", 31, 3},
    {"April", "Apr", 30, 4},
    {"May", "May", 31, 5},
    {"June", "Jun", 30, 6},
    {"July", "Jul", 31, 7},
    {"August", "Aug", 31, 8},
    {"September", "Sep", 30, 9},
    {"October", "Oct", 31, 10},
    {"November", "Nov", 30, 11},
    {"December", "Dec", 31, 12}
};

int days(char * m);
int main(void)
{
    char input[20];
    int daytotal;

    printf("Enter the name of a month: ");
    while (gets(input) != NULL && input[0] != '\0')
    {
        daytotal = days(input);
        if (daytotal > 0)
            printf("There are %d days through %s.\n", daytotal, input);
        else
            printf("%s is not valid input.\n", input);
        printf("Next month (empty line to quit): ");
    }
    puts("bye");

    return 0;
}

int days(char * m)
{
    int total = 0;
    int mon_num = 0;
    int i;
    if (m[0] == '\0')
        total = -1;
    else
    {
        m[0] = toupper(m[0]);
        for (i = 1; m[i] != '\0'; i++)
            m[i] = tolower(m[i]);
        for (i = 0; i < 12; i++)
            if (strcmp(m, months[i].name) == 0)
            {
                mon_num = months[i].monumb;
                break;
            }
        if (mon_num == 0)
            total = -1;
        else
            for (i = 0; i < mon_num; i++)
                total +=months[i].days;
    }
    return total;
}
```

PE 14-3

```
/* pe14-3.c */
#include <stdio.h>
#include <string.h>
#define MAXTITL   40
#define MAXAUTL   40
#define MAXBKS   100            /* maximum number of books  */
struct book {                   /* set up book template     */
    char title[MAXTITL];
    char author[MAXAUTL];
    float value;
};

void sortt(struct book * pb[], int n);
void sortv(struct book * pb[], int n);

int main(void)
{
     struct book library[MAXBKS]; /* array of book structures */
     struct book * pbk[MAXBKS];   /* pointers for sorting     */
     int count = 0;
     int index;

     printf("Please enter the book title.\n");
     printf("Press [enter] at the start of a line to stop.\n");
     while (count < MAXBKS && gets(library[count].title) != NULL
                         && library[count].title[0] != '\0')
     {
          printf("Now enter the author.\n");
          gets(library[count].author);
          printf("Now enter the value.\n");
          scanf("%f", &library[count].value);
          pbk[count] = &library[count];
          count++;
          while (getchar() != '\n')
               continue;                /* clear input line */
          if (count < MAXBKS)
          printf("Enter the next title.\n");
     }
     printf("Here is the list of your books:\n");
     for (index = 0; index < count; index++)
         printf("%s by %s: $%.2f\n", library[index].title,
          library[index].author, library[index].value);

     printf("Here is the list of your books sorted by title:\n");
     sortt(pbk, count);
     for (index = 0; index < count; index++)
         printf("%s by %s: $%.2f\n", pbk[index]->title,
          pbk[index]->author, pbk[index]->value);
     sortv(pbk, count);
     printf("Here is the list of your books sorted by value:\n");
     for (index = 0; index < count; index++)
         printf("%s by %s: $%.2f\n", pbk[index]->title,
          pbk[index]->author, pbk[index]->value);

     return 0;
}

void sortt(struct book * pb[], int n)
{
   int top, search;
   struct book * temp;

   for (top = 0; top < n -1; top++)
       for (search = top + 1; search < n; search++)
            if (strcmp(pb[search]->title, pb[top]->title) < 0)
            {
                 temp = pb[search];
                 pb[search] = pb[top];
                 pb[top] = temp;
            }
}

void sortv(struct book * pb[], int n)
{
   int top, search;
   struct book * temp;

   for (top = 0; top < n -1; top++)
       for (search = top + 1; search < n; search++)
            if (pb[search]->value < pb[top]->value)
            {
                 temp = pb[search];
                 pb[search] = pb[top];
                 pb[top] = temp;
            }
}
```

PE 14-5

```
    /* pe14-5.c */
#include <stdio.h>
#include <string.h>
#define LEN 14
#define CSIZE 4
#define SCORES 3
struct name {
    char first[LEN];
    char last[LEN];
};
```

```
struct student {
    struct name person;
    float scores[SCORES];
    float mean;
};
void get_scores(struct student ar[], int lim);
void find_means(struct student ar[], int lim);
void show_class(const struct student ar[], int lim);
void show_ave(const struct student ar[], int lim);

int main(void)
{
    struct student class[CSIZE] ={
        { "Flip", "Snide"},
        { "Clare", "Voyans"},
        { "Bingo", "Higgs"},
        { "Fawn", "Hunter"}
    };

    get_scores(class, CSIZE);
    find_means(class, CSIZE);
    show_class(class, CSIZE);
    show_ave(class, CSIZE);
    return 0;
}

void get_scores(struct student ar[], int lim)
{
    int i,j;
    for (i = 0; i < lim; i++)
    {
        printf ("Please enter %d scores for %s %s:\n", SCORES,
            ar[i].person.first, ar[i].person.last);
        for (j = 0; j < SCORES; j++)
        {
            while (scanf("%f", &ar[i].scores[j]) != 1)
            {
                scanf("%*s");
                puts("Please use numeric input.");
            }
        }
    }
}

void find_means(struct student ar[], int lim)
{
    int i, j;
    float sum;

    for (i = 0; i < lim; i++)
    {
        for (sum = 0, j = 0; j < SCORES; j++)
            sum += ar[i].scores[j];
        ar[i].mean = sum / SCORES;
    }
}
```

```
void show_class(const struct student ar[], int lim)
{
    int i, j;
    char wholename[2*LEN];

    for (i = 0; i < lim; i++)
    {
        strcpy(wholename, ar[i].person.first);
         strcat(wholename, " ");
         strcat(wholename, ar[i].person.last);
          printf("%27s: ", wholename);
          for (j = 0; j < SCORES; j++)
            printf("%6.1f ", ar[i].scores[j]);
        printf(" Average = %5.2f\n", ar[i].mean);
    }
}

void show_ave (const struct student ar[], int lim)
{
    int i, j;
    float total;

    printf("\n%27s: ", "QUIZ AVERAGES");
    for (j = 0; j < SCORES; j++)
    {
        for (total = 0, i = 0; i < lim; i++)
            total += ar[i].scores[j];
        printf("%6.2f ", total / lim);
    }
    putchar('\n');
}
```

PE 14-7

```
/* pe14-7.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAXTITL     40
#define MAXAUTL     40
#define MAXBKS      10          /* maximum number of books */
#define CONTINUE    0
#define DONE        1
#define YES         1
#define NO          0
struct book {                   /* set up book template    */
    char title[MAXTITL];
    char author[MAXAUTL];
    float value;
    int delete;
};

int getlet(const char * s);
int getbook(struct book * pb);
void update(struct book * item);
```

```
int main(void)
{
     struct book library[MAXBKS]; /* array of structures     */
     int count = 0;
     int deleted = 0;
     int index, filecount, open;
     FILE * pbooks;
     int size = sizeof (struct book);

     if ((pbooks = fopen("book.dat", "r")) != NULL)
     {
         while (count < MAXBKS &&  fread(&library[count], size,
                     1, pbooks) == 1)
         {
             if (count == 0)
                 puts("Current contents of book.dat:");
             printf("%s by %s: $%.2f\n",library[count].title,
                 library[count].author, library[count].value);
             printf("Do you wish to change or delete this entry?<y/n> ");
             if (getlet("yn") == 'y')
             {
                 printf("Enter c to change, d to delete entry: ");
                 if (getlet("cd") == 'd')
                 {
                     library[count].delete = YES;
                     deleted++;
                     puts("Entry marked for deletion.");
                 }
                 else
                     update(&library[count]);
             }
             count++;
         }
         fclose(pbooks);
     }
     filecount = count - deleted;
     if (count == MAXBKS)
     {
         fputs("The book.dat file is full.", stderr);
         exit(2);
     }
     puts("Please add new book titles.");
     puts("Press [enter] at the start of a line to stop.");
     open = 0;
     while (filecount < MAXBKS)
     {
        if (filecount < count)
        {
            while (library[open].delete == NO)
                open++;
            if (getbook(&library[open]) == DONE)
                break;
        }
        else if (getbook(&library[filecount]) == DONE)
            break;
        filecount++;
```

```
         if (filecount < MAXBKS)
             puts("Enter the next book title.");
      }
      puts("Here is the list of your books:");
      for (index = 0; index < filecount; index++)
          if (library[index].delete == NO)
             printf("%s by %s: $%.2f\n",library[index].title,
                 library[index].author, library[index].value);
      if ((pbooks = fopen("book.dat", "w")) == NULL)
      {
          fputs("Can't open book.dat file for output\n",stderr);
          exit(1);
      }
      for (index = 0; index < filecount; index++)
          if (library[index].delete == NO)
             fwrite(&library[index], size, 1, pbooks);
      fclose(pbooks);
      puts("Done!");

      return 0;
}

int getlet(const char * s)
{
    char c;

    c = getchar();
    while (strchr(s, c) == NULL)
    {
        printf ("Enter a character in the list %s\n", s);
        while( getchar() != '\n')
            continue;
        c = getchar();
    }
    while (getchar() != '\n')
        continue;

    return c;
}

int getbook(struct book * pb)
{
    int status = CONTINUE;
    if (gets(pb->title) == NULL || pb->title[0] == '\0')
        status = DONE;
    else
    {
        printf ("Now enter the author: ");
        gets (pb->author);
        printf ("Now enter the value: ");
        while (scanf("%f", &pb->value ) != 1)
        {
            puts("Please use numeric input");
            scanf("%*s");
        }
        while (getchar() != '\n')
            continue; /*clear input line */
        pb->delete = NO;
    }
    return status;
}

void update(struct book * item)
{
    struct book copy;
    char c;

    copy = *item;
    puts("Enter the letter that indicates your choice:");
    puts("t) modify title     a) modify author");
    puts("v) modify value     s) quit, saving changes");
    puts("q) quit, ignore changes");
    while ( (c = getlet("tavsq")) != 's' && c != 'q')
    {
        switch ( c )
        {
            case 't' : puts("Enter new title: ");
                       gets (copy.title);
                       break;
            case 'a' : puts("Enter new author: ");
                       gets (copy.author);
                       break;
            case 'v' : puts("Enter new value: ");
                       while (scanf("%f", &copy.value) != 1)
                       {
                           puts ("Enter a numeric value: ");
                           scanf("%*s");
                       }
                       while( getchar() != '\n')
                           continue;
                       break;
        }
        puts("t) modify title     a) modify author");
        puts("v) modify value     s) quit, saving changes");
        puts("q) quit, ignore changes");
    }
    if (c == 's')
        *item = copy;
}
```

PE 14-8

```
/* pe14-8.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define LEN         14
#define SEATS       12
#define EMPTY       0
#define TAKEN       1
#define CONTINUE    1
```

```
#define DONE        0

struct planestats {
    int seat_id;
    int status;
    char last[LEN];
    char first[LEN];
};

int getmenu(void);
int getlet(const char *);
int openings(const struct planestats [], int);
void show_empties(const struct planestats [], int);
void list_assign(struct planestats *[], int);
void assign_seat(struct planestats [], int);
void delete_seat(struct planestats [], int);
void show_seats(const struct planestats [], int);
void sort(struct planestats *[], int);
void makelist(const struct planestats [], char *, int);

int main(void)
{
    struct planestats plane_1[SEATS], *ps[SEATS];
    int choice;
    int i;
    FILE *fp;
    size_t size = sizeof(struct planestats);

    for ( i = 0; i < SEATS; i++)
        ps[i] = &plane_1[i];
    if ((fp = fopen("air.dat", "rb")) == NULL )
        for (i = 0; i < SEATS; i++)
        {
            plane_1[i].status = EMPTY;
            plane_1[i].seat_id = i + 1;
        }
    else
    {
        fread(plane_1, size, SEATS, fp);
        fclose(fp);
    }
    while ( (choice = getmenu() ) != 'q')
    {
        switch (choice)
        {
            case 'o' :  printf ("There are %d empty seats.\n",
                        penings(plane_1, SEATS));
                        break;
            case 'e' :  show_empties(plane_1, SEATS);
                        break;
            case 'l' :  list_assign(ps, SEATS);
                        break;
            case 'a' :  assign_seat(plane_1, SEATS);
                        break;
            case 'd' :  delete_seat(plane_1, SEATS);
                        break;
            default  :  puts("Switch trouble");
```

```
                        break;
            }
        }
        if((fp = fopen("air.dat", "wb")) == NULL )
            puts("Can't save data to file.");
        else
        {
            fwrite(plane_1, size, SEATS, fp);
            fclose(fp);
        }
        puts("Bye from Colossus Airlines!");
        return 0;
}

#define CHOICES 6
int getmenu(void)
{
    const char *descript[CHOICES] = {
        "Show number of empty seats",
        "Show list of empty seats",
        "Show alphabetical list of seat assignments",
        "Assign a customer to a seat",
        "Delete a seat assignment",
        "Quit"
    };
    const char labels[CHOICES + 1] = "oeladq";
    int i;

    puts("To choose a function, enter its letter label");
    for (i = 0; i < CHOICES; i++)
        printf("%c) %s\n", labels[i], descript[i]);
    return getlet(labels);
}

int getlet(const char * s)
{
    char c;

    c = getchar();
    while (strchr(s, c) == NULL)
    {
        printf ("Enter a character in the list %s\n", s);
        while( getchar() != '\n')
            continue;
        c = getchar();
    }
    while (getchar() != '\n')
        continue;

    return c;
}

int openings(const struct planestats pl[], int n)
{
    int count = 0;
    int seat;
    for (seat = 0; seat    < n; seat++)
        if (pl[seat].status == EMPTY)
            count++;
    return count;
}

void show_empties(const struct planestats pl[], int n)
{
    int seat;
    char seating[3* SEATS];

    if ( openings(pl,n) == 0)
        puts("All seats are assigned");
    else
    {
        puts("The following seats are available:");
        makelist(pl, seating, EMPTY);
        puts (seating) ;
    }
}

void makelist(const struct planestats pl[], char * str, int kind)
{
    int seat;
    char temp[LEN];

    str[0] = '\0';
    for (seat = 0; seat    < SEATS; seat++)
        if (pl[seat].status == kind)
        {
            sprintf(temp," %d", pl[seat].seat_id);
            strcat(str, temp);
        }
}

void list_assign(struct planestats *ps[], int n)
{
    int i;
    if (openings(*ps, n) == SEATS)
        puts("All seats are empty.");
    else
    {
        sort(ps, n);
        for(i = 0; i < SEATS; i++)
            if ( ps[i]->status == TAKEN )
                printf ("Seat %d: %s, %s\n",
                    ps[i]->seat_id, ps[i]->last, ps[i]->first);
    }
}

void assign_seat(struct planestats pl[], int n)
{
    char list[3    * SEATS];
    int seat, loop;

    if (openings(pl,n) == 0)
        puts("All seats are assigned.");
    else
    {
        makelist(pl,list, EMPTY);
        puts("Which seat do you want? Choose from this list:");
        puts (list) ;
        do
        {
            while( scanf("%d", &seat) != 1)
            {
                scanf("%*s");
                puts("Enter a number from this list:");
                puts (list) ;
            }
            if (seat < 1 || seat > SEATS ||
                pl[seat-1].status == TAKEN)
            {
                puts("Enter a number from this list:");
                puts (list) ;
                loop = CONTINUE;
            }
            else
                loop = DONE;
        } while (loop == CONTINUE);
        while (getchar() != '\n')
            continue;
        puts("Enter first name:");
        gets (pl[seat - 1].first);
        puts("Enter last name:");
        gets (pl[seat - 1].last);
        printf("%s %s assigned to seat %d.\n",
        pl[seat - 1].first, pl[seat - 1].last, seat);
        puts("Enter a to accept assignment, c to cancel it.");
        if (getlet("ac") == 'a')
        {
            pl[seat - 1].status = TAKEN;
            puts("Passenger assigned to seat.");
        }
        else
            puts("Passenger not assigned.");
    }
}

void delete_seat(struct planestats pl[], int n)
{
    int seat, loop;
    char list[3    * SEATS];

    if (openings(pl, n) == SEATS)
        puts("All seats already are empty.");
    else
    {
        show_seats(pl, n);
        makelist(pl, list, TAKEN);
        puts("Enter the number of the seat to be cancelled:");
        do
        {
            while( scanf("%d", &seat) != 1)
            {
                scanf("%*s");
                puts("Enter a number from this list:");
                puts (list) ;
            }
            if (seat < 1 || seat > SEATS ||
                 pl[seat-1].status == EMPTY)
            {
                puts("Enter a number from this list:");
                puts (list) ;
                loop = CONTINUE;
            }
            else
                loop = DONE;
        } while (loop == CONTINUE);
        while (getchar() != '\n')
            continue;
        printf("%s %s to be canceled for seat %d.\n",
            pl[seat - 1].first, pl[seat - 1].last, seat);
        puts("Enter d to delete assignment, a to abort.");
        if ( getlet("da") == 'd')
        {
            pl[seat - 1].status = EMPTY;
            puts ("Passenger dropped.");
        }
        else
            puts("Passenger retained.");
    }
}

void show_seats(const struct planestats pl[], int n)
{
    int i;

    puts("Seats currently taken:");
    for (i = 0; i < SEATS; i++)
        if (pl[i].status == TAKEN)
            printf("Seat %d: %s, %s\n", pl[i].seat_id,
                pl[i].last, pl[i].first);
}

void sort(struct planestats *array[], int limit)
{
   int top, search;
   struct planestats * temp;

   for (top = 0; top < limit -1; top++)
       for (search = top + 1; search < limit; search++)
            if (strcmp(array[search]->last, array[top]->last) < 0)
            {
                 temp = array[search];
                 array[search] = array[top];
                 array[top] = temp;
            }
}
```

PE 14-10

```
/* pe14-10.c */
/* the tricky part is declaring an array of pointers to functions */
#include <stdio.h>
#include <math.h>

double twice(double x);
double half(double x);
double thrice(double x);
void showmenu(void);
#define NUM 4
int main(void)
{
    double (*pf[NUM])(double)  = {twice, half, thrice, sqrt};
    double val;
    double ans;
    int sel;

    printf("Enter a number (negative to quit): ");
    while (scanf("%lf", &val) && val >= 0)
    {
        showmenu();
        while (scanf("%d", &sel) && sel >= 0 && sel <= 3)
        {
            ans = (*pf[sel])(val);
            printf("answer = %f\n", ans);
            showmenu();
        }
        printf("Enter next number (negative to quit): ");

    }
    puts("bye");
    return 0;
}


void showmenu(void)
{
    puts("Enter one of the following choices:");
    puts("0) double the value        1) halve the value");
    puts("2) triple the value        3) squareroot the value");
    puts("4) next number");
}

double twice(double x) {return 2.0 * x;}
double half(double x) {return x / 2.0;};
double thrice(double x) {return 3.0 * x;}
```

Chapter 15

PE 15-1

```
/* pe15-1.c */
#include <stdio.h>
#include <stdbool.h>  // C99 -- otherwise use int

int bstr_to_dec(const char * str);
bool check_val(const char * str);
int main(void)
{
    char value[8* sizeof (int) + 1];

    printf("Enter a binary number with up to %d digits: ", 8 * sizeof(int));

    while (gets(value) && value[0] != '\0')
    {
        if (!check_val(value))
            puts("A binary number contains just 0s and 1s.");
        else
            printf("%s is %d\n", value, bstr_to_dec(value));
        puts("Enter next value:");
    }

    puts("Done");
    return 0;
}

int bstr_to_dec(const char * str)
{
    int val = 0;

    while (*str != '\0')
        val = 2 * val + (*str++ - '0');
    return val;
}

bool check_val(const char * str)
{
    bool valid = true;

    while (valid && *str != '\0')
    {
        if (*str != '0' && *str != '1')
            valid = false;
        ++str;
    }

    return valid;
}
```

PE 15-2

```
/* pe15-2.c /
#include <stdio.h>
```

```
#include <stdlib.h>
/* #include <console.h> */    /* Macintosh only */

int bstr_to_dec(const char * str);
char * itobs(int, char *);
int main(int argc, char * argv[])
{
    int v1;
    int v2;
    char bstr[8* sizeof (int) + 1];

    /* argc = ccommand(&argv); */   /* Macintosh only */

    if (argc != 3)
    {
        fprintf(stderr, "Usage: %s binarynum1 binarynum2\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    v1 = bstr_to_dec(argv[1]);
    v2 = bstr_to_dec(argv[2]);

    printf("~%s = %s\n", argv[1], itobs(~v1, bstr));
    printf("~%s = %s\n", argv[2], itobs(~v2, bstr));
    printf("%s & %s= %s\n", argv[1], argv[2], itobs(v1 & v2, bstr));
    printf("%s | %s= %s\n", argv[1], argv[2], itobs(v1 | v2, bstr));
    printf("%s ^ %s= %s\n", argv[1], argv[2], itobs(v1 ^ v2, bstr));

    puts("Done");
    return 0;
}

int bstr_to_dec(const char * str)
{
    int val = 0;

    while (*str != '\0')
        val = 2 * val + (*str++ - '0');
    return val;
}

char * itobs(int n, char * ps)
{
    int i;
    static int size = 8 * sizeof(int);

    for (i = size - 1; i >= 0; i--, n >>= 1)
        ps[i] = (01 & n) + '0';
    ps[size] = '\0';

    return ps;
}
```

PE 15-3

```
/* pe15-3.c */
#include <stdio.h>
char * itobs(int, char *);
int onbits(int);
int main(int argc, char * argv[])
{
    int val;
    char bstr[8* sizeof (int) + 1];

    printf("Enter an integer (negative to quit): ");
    while (scanf("%d", &val) && val >= 0)
    {
        printf ("%d (%s) has %d bits on.\n", val,
                  itobs(val, bstr), onbits(val));
        printf("Next value: ");
    }

    puts("Done");
    return 0;
}

char * itobs(int n, char * ps)
{
    int i;
    static int size = 8 * sizeof(int);

    for (i = size - 1; i >= 0; i--, n >>= 1)
        ps[i] = (01 & n) + '0';
    ps[size] = '\0';

    return ps;
}

int onbits(int n)
{
    static const int size = 8 * sizeof(int);
    int ct = 0;
    int i;

    for (i = 0; i < size; i++, n >>= 1)
        if ((1 & n) == 1)
            ct++;
    return ct;
}
```

PE 15-5

```
/* pe15-5.c */
#include <stdio.h>
unsigned int rotate_l(unsigned int, unsigned int);
char * itobs(int, char *);

int main(void)
{
    unsigned int val;
    unsigned int rot;
    unsigned int places;
    char bstr1[8* sizeof (int) + 1];
    char bstr2[8* sizeof (int) + 1];

    printf("Enter an integer (0 to quit): ");
    while (scanf("%ud", &val) && val > 0)
    {
        printf("Enter the number of bits to be rotated: \n");
        scanf("%ul", &places);
        rot = rotate_l(val, places);
        itobs(val, bstr1);
        itobs(rot, bstr2);
        printf ("%u rotated is %u.\n", val, rot );
        printf("%s rotated is %s.\n", bstr1, bstr2);
        printf("Next value: ");
    }

    puts("Done");
    return 0;
}

unsigned int rotate_l(unsigned int n, unsigned int b)
{
    static const int size = 8 * sizeof(int);
    unsigned int overflow;

    b %= size;  /* keep b a valid value */

    overflow = n >> (size - b);  /* save bits that are shifted out */
    return (n << b) | overflow;
}
char * itobs(int n, char * ps)
{
    int i;
    static int size = 8 * sizeof(int);

    for (i = size - 1; i >= 0; i--, n >>= 1)
        ps[i] = (01 & n) + '0';
    ps[size] = '\0';

    return ps;
}
```

PE 15-7

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

#define ID_MASK     0xFF
#define SIZE_MASK   0x7F00
#define LEFT        0x00000
#define CENTER      0x08000
#define RIGHT       0x10000
```

```
#define ALIGN_MASK  0x18000
#define REGULAR     0x00000
#define BOLD        0x20000
#define ITALIC      0x40000
#define UNDERLINE   0x80000
#define STYLE_MASK  0xE0000
#define SIZE_SHIFT  8

typedef unsigned long font;


char do_menu(font * f);
char get_choice(const char *);
void show_menu(void);
void show_font(font f);
void eatline(void);
void get_id(font * f);
void get_size(font * f);
void get_align(font * f);

int main(void)
{
    font sample = 1 | (12 <<SIZE_SHIFT) | LEFT | ITALIC;

    while (do_menu(&sample) != 'q')
        continue;
    puts("Bye!");
    return 0;
}

char do_menu(font * f)
{
    char response;

    show_font(*f);
    show_menu();
    response = get_choice("fsabiuq");
    switch(response)
    {
        case 'f' : get_id(f); break;
        case 's' : get_size(f); break;
        case 'a' : get_align(f); break;
        case 'b' : *f ^= BOLD; break;
        case 'i' : *f ^= ITALIC; break;
        case 'u' : *f ^= UNDERLINE; break;
        case 'q' : break;
        default  : fprintf(stderr, "menu problem\n");
    }

    return response;
}

char get_choice(const char * str)
{
    char ch;

    ch = getchar();
    ch = tolower(ch);
    eatline();
    while (strchr(str, ch) == NULL)
    {
        printf("Please enter one of the following: %s\n",
               str);
        ch = tolower(getchar());
        eatline();
    }
    return ch;
}

void eatline(void)
{
    while (getchar() != '\n')
        continue;
}


void show_menu(void)
{
    puts("f)change font    s)change size    a)change alignment");
    puts("b)toggle bold    i)toggle italic  u)toggle underline");
    puts("q)quit");
}


void show_font(font f)
{
    printf("\n%4s %4s %9s %3s %3s %3s\n",
           "ID", "SIZE", "ALIGNMENT", "B", "I", "U");
    printf("%4d %4d", f & ID_MASK, (f & SIZE_MASK) >> SIZE_SHIFT);
    switch(f & ALIGN_MASK)
    {
        case LEFT   : printf("%7s", "left"); break;
        case RIGHT  : printf("%7s", "right"); break;
        case CENTER : printf("%7s", "center"); break;
        default     : printf("%7s", "unknown"); break;
    }
    printf("%8s %3s %3s\n\n", (f & BOLD) == BOLD? "on" : "off",
           (f & ITALIC) == ITALIC ? "on" : "off",
           (f & UNDERLINE) == UNDERLINE ? "on" : "off");
}

void get_id(font * f)
{
    int id;

    printf("Enter font ID (0-255): ");
    scanf("%d", &id);
    id = id & ID_MASK;
    *f |= id;
    eatline();
}

void get_size(font * f)
{
    int size;

    printf("Enter font size (0-127): ");
    scanf("%d", &size);
    *f |= (size << SIZE_SHIFT) & SIZE_MASK;
    eatline();
}

void get_align(font * f)
{
    puts("Select alignment:");
    puts("l)left   c)center   r)right");
    switch (get_choice("lcr"))
    {
        case 'l' : *f &= ~ALIGN_MASK; *f |= LEFT; break;
        case 'c' : *f &= ~ALIGN_MASK; *f |= CENTER; break;
          case 'r' : *f &= ~ALIGN_MASK; *f |= RIGHT; break;
          default  : fprintf(stderr, "alignment problem\n");
    }
}
```

Chapter 16

PE 16-2

```
/* pe16-2.c */
#include <stdio.h>
#define HMEAN(X,Y) (2.0 * (X) *(Y) / ((X) + (Y)))
int main(void)
{
    double x, y, ans;

    while (scanf("%lf %lf", &x, &y) == 2)
    {
        ans = HMEAN(x,y);
        printf("%g = harmonic mean of %g %g.\n", ans, x, y);
        ans = HMEAN(x - y, x +y);
        printf("%g = harmonic mean of %g %g.\n", ans, x - y, x + y);
    }
    puts("Bye");

    return 0;
}
```

PE 16-3

```
/* pe16-3.c */
#include <stdio.h>
#include <math.h>

struct polar {
    double r;
    double theta;   /* angle in degrees */
};

struct rect {
    double x;
    double y;
};

struct rect p_to_r(const struct polar * ppol);

int main(void)
{

    struct polar input;
    struct rect answer;

    while (scanf("%lf %lf", &input.r, &input.theta) == 2)
    {
        answer = p_to_r(&input);
        printf("polar coord: %g %f\n",input.r, input.theta);
        printf("rectangular coord: %g %f\n",answer.x, answer.y);
    }
    puts("Bye");

    return 0;
}

struct rect p_to_r(const struct polar * ppol)
{
    static const double deg_rad = 3.141592654 / 180.0;
    struct rect res;
    double ang = deg_rad * ppol->theta;  /* convert degrees to radians */

    res.x = ppol->r * sin(ang);
    res.y = ppol->r * cos(ang);

    return res;
}
```

PE 16-5

```
/* pe16-5.c */
#include <stdio.h>
#include <time.h>

void wait(double t);

void random_pick(int ar[], int arsize, int picks);
#define SPOTS 51
#define PICKS 6
int main()
{
    int lotto[SPOTS];
    int i;
    char ch;

    for (i = 0; i < SPOTS; i++)
        lotto[i] = i + 1;

    do {
        random_pick(lotto, SPOTS, PICKS);
        printf ("Again? <y/n> ");
        ch = getchar();
        while (getchar() != '\n')
            continue;
    } while (ch == 'y' || ch == 'Y');

    puts ("Done");
    return 0;
}

void random_pick(int ar[], int arsize, int picks)
{
    int i, index, temp;

    srand(time(0));
    if (picks > arsize)
    {
        fputs("Number of picks > array size\n", stderr);
        fputs("Setting picks = array size\n", stderr);
        picks = arsize;
    }
    for (i = 0; i < picks; i++)
    {
        index = rand() % (arsize - 1); /* pick a random element     */
        temp = ar[index];
        printf ("%2d ", temp);         /* display it                */
        if (i % 20 == 19)
            putchar('\n');
        ar[index] = ar[arsize - 1];    /* swap it with last element */
        ar[arsize - 1] = temp;
        arsize--;                      /* exclude end from search   */
    }
    if (i % 20 != 0)
        putchar('\n');
}
```

PE 16-7

```
// pe16-7.c.-- using a variadic function
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
void show_array(const double ar[], int n);
double * new_d_array(int n, ...);
int main()
{
    double * p1;
    double * p2;
    p1 = new_d_array(5, 1.2, 2.3, 3.4, 4.5, 5.6);
    p2 = new_d_array(4, 100.0, 20.00, 8.08, -1890.0);
    show_array(p1, 5);
    show_array(p2, 4);
    free(p1);
    free(p2);

    return 0;
}

void show_array(const double ar[], int n)
{
    int i;

    for (i = 0; i < n; i++)
        printf("%g ", ar[i]);
    putchar('\n');
}

double * new_d_array(int n, ...)
{
    va_list ap;
    int i;
    double * pt;

    va_start(ap, n);
    pt = (double *) malloc(n * sizeof(double));
    for (i = 0; i< n; i++)
        pt[i] = va_arg(ap, double);
    va_end(ap);
    return pt;
}
```

Chapter 17

PE 17-1

```
/* pe17-1a.c  recursive solution */
#include <stdio.h>
#include <stdlib.h>      /* has the malloc prototype      */
#include <string.h>      /* has the strcpy prototype      */
#define TSIZE    45      /* size of array to hold title   */
struct film {
   char title[TSIZE];
   int rating;
   struct film * next;   /* points to next struct in list */
};

void show_rec(const struct film * pf); /* recursive function */
int main(void)
{
   struct film * head = NULL;
   struct film * prev, * current;
   char input[TSIZE];

   puts("Enter first movie title:");
   while (gets(input) != NULL && input[0] != '\0')
   {
      current = (struct film *) malloc(sizeof(struct film));
      if (head == NULL)          /* first structure       */
         head = current;
      else                       /* subsequent structures */
         prev->next = current;
      current->next = NULL;
      strcpy(current->title, input);
      puts("Enter your rating <0-10>:");
      scanf("%d", &current->rating);
      while(getchar() != '\n')
         continue;
      puts("Enter next movie title (empty line to stop):");
      prev = current;
   }
   if (head == NULL)
      printf("No data entered. ");
   else
      printf ("Here is the movie list:\n");
   current = head;
   while (current != NULL)
   {
      printf("Movie: %s  Rating: %d\n", current->title, current->rating);
      current = current->next;
   }
   if (head != NULL)
   {
         printf("\nHere is the list in reverse order:\n");
      show_rec(head);
   }
   printf("Bye!\n");
   return 0;
}

void show_rec(const struct film * pf)
{
    if (pf->next != NULL)
        show_rec(pf->next);
    printf("Movie: %s  Rating: %d\n", pf->title, pf->rating);
}


/* pe17-1b.c -- double-link solution */
#include <stdio.h>
#include <stdlib.h>      /* has the malloc prototype      */
#include <string.h>      /* has the strcpy prototype      */
#define TSIZE    45      /* size of array to hold title   */
struct film {
   char title[TSIZE];
   int rating;
   struct film * next;   /* points to next struct in list */
   struct film * prev;   /* points to previous struct     */
```

```
};

int main(void)
{
    struct film * head = NULL;
    struct film * prev, * current;
    char input[TSIZE];

    puts("Enter first movie title:");
    while (gets(input) != NULL && input[0] != '\0')
    {
        current = (struct film *) malloc(sizeof(struct film));
        if (head == NULL)         /* first structure       */
        {
            head = current;
            head->prev = NULL;
        }
        else                      /* subsequent structures */
        {
            prev->next = current;
            current->prev = prev;
        }
        current->next = NULL;
        strcpy(current->title, input);
        puts("Enter your rating <0-10>:");
        scanf("%d", &current->rating);
        while(getchar() != '\n')
            continue;
        puts("Enter next movie title (empty line to stop):");
        prev = current;
    }
    if (head == NULL)
        printf("No data entered. ");
    else
        printf ("Here is the movie list:\n");
    current = head;
    while (current != NULL)
    {
        printf("Movie: %s  Rating: %d\n", current->title, current->rating);
        prev = current;
        current = current->next;
    }
    if (head != NULL)
    {
        printf("\nHere is the list in reverse order:\n");
        current = prev;
        while (current != NULL)
        {
            printf("Movie: %s  Rating: %d\n", current->title,
                    current->rating);
            current = current->prev;
        }
    }
    printf("Bye!\n");
    return 0;
}
```

PE 17-3

```
/* list17-3.h -- header file for a simple list type */
#ifndef LIST_H_
#define LIST_H_
#include <stdbool.h> /* C99 -- else define bool with enum */

/* program-specific declarations */

#define TSIZE      45    /* size of array to hold title  */
struct film
{
   char title[TSIZE];
   int rating;
};

/* general type definitions */

typedef struct film Item;

typedef struct node
{
   Item item;
   struct node * next;
} Node;

#define MAXSIZE 100
typedef struct list
{
    Item entries[MAXSIZE];    /* array of items */
    int items;                /* number of items */
} List;


/* function prototypes */

/* operation:        initialize a list                          */
/* preconditions:    plist points to a list                     */
/* postconditions:   the list is initialized to empty           */
void InitializeList(List * plist);

/* operation:        determine if list is empty                 */
/* preconditions:    l is an initialized list                   */
/* postconditions:   function returns true if list is empty     */
/*                   and returns false otherwise                */
bool ListIsEmpty(const List * plist);

/* operation:        determine if list is full                  */
/* preconditions:    l is an initialized list                   */
/* postconditions:   function returns true if list is full      */
/*                   and returns false otherwise                */
bool ListIsFull(const List * plist);

/* operation:        determine number of items in list          */
/* preconditions:    l is an initialized list                   */
/* postconditions:   function returns number of items in list   */
```

```
unsigned int ListItemCount(const List * plist);

/* operation:        add item to end of list                    */
/* preconditions:    item is an item to be added to list        */
/*                   plist points to an initialized list        */
/* postconditions:   if possible, function adds item to end     */
/*                   of list and returns true; otherwise the    */
/*                   function returns false                     */
bool AddItem(Item item, List * plist);

/* operation:        apply a function to each item in list      */
/* preconditions:    l is an initialized list                   */
/*                   pfun points to a function that takes an    */
/*                   Item argument and has no return value      */
/* postcondition:    the function pointed to by pfun is         */
/*                   executed once for each item in the list    */
void Traverse (const List * plist, void (* pfun)(Item item) );

/* operation:        free allocated memory, if any              */
/*                   plist points to an initialized list        */
/* postconditions:   any memory allocated for the list is freed */
/*                   and the list is set to empty               */
void EmptyTheList(List * plist);

#endif


/* pe17-3a.c -- a copy of films3.c  */
/* compile with pe17-3b.c                        */
#include <stdio.h>
#include <stdlib.h>    /* prototype for exit() */
#include "list17-3.h"  /* defines List, Item   */
void showmovies(Item item);

int main(void)
{
    List movies;
    Item temp;


/* initialize       */
    InitializeList(&movies);
    if (ListIsFull(&movies))
    {
        fprintf(stderr,"No memory available! Bye!\n");
        exit(1);
    }

/* gather and store */
    puts("Enter first movie title:");
    while (gets(temp.title) != NULL && temp.title[0] != '\0')
    {
        puts("Enter your rating <0-10>:");
        scanf("%d", &temp.rating);
        while(getchar() != '\n')
            continue;
        if (AddItem(temp, &movies)==false)
        {
            fprintf(stderr,"Problem allocating memory\n");
            break;
        }
        if (ListIsFull(&movies))
        {
            puts("The list is now full.");
            break;
        }
        puts("Enter next movie title (empty line to stop):");
    }

/* display          */
    if (ListIsEmpty(&movies))
        printf("No data entered. ");
    else
    {
        printf ("Here is the movie list:\n");
        Traverse(&movies, showmovies);
    }
    printf("You entered %d movies.\n", ListItemCount(&movies));


/* clean up         */
    EmptyTheList(&movies);
    printf("Bye!\n");

    return 0;
}

void showmovies(Item item)
{
    printf("Movie: %s  Rating: %d\n", item.title,
            item.rating);
}


/* pe17-3b.c -- revised list.c -- functions supporting list operations */
#include <stdio.h>
#include <stdlib.h>
#include "list17-3.h"

/* interface functions */
/* set the list to empty   */
void InitializeList(List * plist)
{
   plist->items = 0;
}

/* returns true if list is empty */
bool ListIsEmpty(const List * plist)
{
   if (plist->items == 0)
      return true;
   else
```

```
      return false;
}

/* returns true if list is full */
bool ListIsFull(const List * plist)
{
       if (plist->items == MAXSIZE)
           return true;
       else
           return false;
}

/* returns number of items in list */
unsigned int ListItemCount(const List * plist)
{
     return plist->items;
}

/* adds item to list */
/* assumes = operator defined for type Item */
bool AddItem(Item item, List * plist)
{
    if (plist->items == MAXSIZE)
        return false;
    else
    {
        plist->entries[plist->items++] = item;
        return true;
    }
}

/* visit each node and execute function pointed to by pfun */
void Traverse (const List * plist, void (* pfun)(Item item) )
{
   int i;

   for (i = 0; i < plist->items; i++)
      (*pfun)(plist->entries[i]);   /* apply function to item in list */
}


/* malloc() not used, nothing need be deallocated */
/* set items member to 0                          */
void EmptyTheList(List * plist)
{
      plist->items = 0;
}
```

PE 17-5

```
/* pe17-5.h --header file for a stack type */

#ifndef STACK_H_
#define STACK_H_
#include <stdbool.h>  /* C99 */
```

```
/* enum bool {false, true}; */  /* pre-C99*/

/* INSERT ITEM TYPE HERE */
/* FOR EXAMPLE, typedef int Item; */

typedef char  Item;

#define MAXSTACK 100

typedef struct stack

{
    Item items[MAXSTACK];   /* holds info                */
    int top;                /* index of first empty slot */
} Stack;

/* operation:       initialize the stack                */
/* precondition:    ps points to a stack                */
/* postcondition:   stack is initialized to being empty */
void InitializeStack(Stack * ps);

/* operation:       check if stack is full                     */
/* precondition:    ps points to previously initialized stack  */
/* postcondition:   returns True if stack is full, else False  */
bool FullStack(const Stack * ps);

/* operation:       check if stack is empty                    */
/* precondition:    ps points to previously initialized stack  */
/* postcondition:   returns True if stack is empty, else False */
bool EmptyStack(const Stack *ps);

/* operation:       push item onto top of stack                */
/* precondition:    ps points to previously initialized stack  */
/*                  item is to be placed on top of stack       */
/* postcondition:   if stack is not empty, item is placed at   */
/*                  top of stack and function returns          */
/*                  True; otherwise, stack is unchanged and    */
/*                  function returns False                     */
bool Push(Item item, Stack * ps);

/* operation:       remove item from top of stack              */
/* precondition:    ps points to previously initialized stack  */
/* postcondition:   if stack is not empty, item at top of      */
/*                  stack is copied to *pitem and deleted from */
/*                  stack, and function returns True; if the   */
/*                  operation empties the stack, the stack is  */
/*                  reset to empty. If the stack is empty to   */
/*                  begin with, stack is unchanged and the     */
/*                  function returns False                     */
bool Pop(Item *pitem, Stack * ps);

#endif

/* pe17-5a.c */
#include <stdio.h>
#include "pe17-5.h"
#define SLEN 81
```

```
int main(void)
{
    Stack stch;
    char temp[SLEN];
    int i;
    char ch;

    InitializeStack(&stch);
    printf("Enter a line (an empty line to quit): \n");
    while (gets(temp) && temp[0] != '\0')
    {
        i = 0;
        while (temp[i] != '\0' && !FullStack(&stch))
            Push(temp[i++], &stch);

        while (!EmptyStack(&stch))
        {
            Pop(&ch, &stch);
            putchar(ch);
        }
        putchar('\n');
        printf("Enter next line (empty line to quit): ");
    }
    puts("Done!");

   return 0;
}


/* pe17-5b.c -- stack operations */
#include <stdio.h>
#include <stdlib.h>
#include "pe17-5.h"

void InitializeStack(Stack * ps)
{
    ps->top = 0;
}

bool FullStack(const Stack * ps)
{
    return ps->top == MAXSTACK;
}

bool EmptyStack(const Stack *ps)
{
    return ps->top == 0;
}

bool Push(Item item, Stack * ps)
{
    if (ps->top == MAXSTACK)
        return false;
    else
    {
        ps->items[ps->top++] = item;
        return true;
    }
}

bool Pop(Item *pitem, Stack * ps)
{
    if (ps->top == 0)
        return false;
    else
    {
        ps->top--;
        *pitem = ps->items[ps->top];
        return true;
    }
}
```

PE 17-7

```
/* tree.h -- binary search tree */
/*           no duplicate items are allowed in this tree */
#ifndef _TREE_H_
#define _TREE_H_
#include <stdbool.h>  /* C99 */
/* enum bool {false, true}; */  /* pre-C99*/
#define SLEN 81

/* redefine Item as appropriate */
typedef struct item
{
    char wrd[SLEN];
    int count;
} Item;

#define MAXITEMS 100

typedef struct node
{
    Item item;
    struct node * left;    /* pointer to right branch  */
    struct node * right;   /* pointer to left branch   */
} Node;

typedef struct tree
{
    Node * root;           /* pointer to root of tree  */
    int size;              /* number of items in tree  */
} Tree;

/* function prototypes */

/* operation:      initialize a tree to empty          */
/* preconditions:  ptree points to a tree              */
/* postconditions: the tree is initialized to empty    */
void InitializeTree(Tree * ptree);

/* operation:      determine if tree is empty          */
/* preconditions:  ptree points to a tree               */
/* postconditions: function returns true if tree is    */
/*                 empty and returns false otherwise   */
bool TreeIsEmpty(const Tree * ptree);

/* operation:      determine if tree is full            */
/* preconditions:  ptree points to a tree               */
/* postconditions: function returns true if tree is    */
/*                 full and returns false otherwise    */
bool TreeIsFull(const Tree * ptree);

/* operation:      determine number of items in tree   */
/* preconditions:  ptree points to a tree               */
/* postconditions: function returns number of items in */
/*                 tree                                 */
int TreeItemCount(const Tree * ptree);

/* operation:      add an item to a tree                */
/* preconditions:  pi is address of item to be added   */
/*                 ptree points to an initialized tree */
/* postconditions: if possible, function adds item to  */
/*                 tree and returns true; otherwise,    */
/*                 the function returns false          */
bool AddItem(const Item * pi, Tree * ptree);

/* operation:      find an item in a tree               */
/* preconditions:  pi points to an item                 */
/*                 ptree points to an initialized tree */
/* postconditions: function returns true if item is in */
/*                 tree and returns false otherwise    */
bool InTree(const Item * pi, const Tree * ptree);

/* operation:      delete an item from a tree           */
/* preconditions:  pi is address of item to be deleted */
/*                 ptree points to an initialized tree */
/* postconditions: if possible, function deletes item  */
/*                 from tree and returns true;         */
/*                 otherwise, the function returns false*/
bool DeleteItem(const Item * pi, Tree * ptree);

/* operation:      apply a function to each item in    */
/*                 the tree                             */
/* preconditions:  ptree points to a tree               */
/*                 pfun points to a function that takes*/
/*                 an Item argument and has no return  */
/*                 value                                */
/* postcondition:  the function pointed to by pfun is  */
/*                 executed once for each item in tree */
void Traverse (const Tree * ptree, void (* pfun)(Item item));

/* operation:      delete everything from a tree       */
/* preconditions:  ptree points to an initialized tree */
/* postconditions: tree is empty                        */
void DeleteAll(Tree * ptree);

/* operation:      return address of item in a tree    */
/* preconditions:  pi points to an item                 */
/*                   ptree points to an initialized tree */
/* postconditions: function returns address if item is */
/*                   in tree and returns NULL otherwise  */
const Item * WhereInTree(const Item * pi, const Tree * ptree);

#endif

/* pe17-7a.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#include "pe17-7.h"
void printitem(Item item);
char menu(void);
void showwords (const Tree * pt);
void findword (const Tree * pt);

#define SLEN 81
int main(void)
{
    Tree wordcount;
    FILE * fp;
    char filename[SLEN];
    char word[SLEN];
    Item entry;
    char choice;


    printf ("Enter name of file to be processed: \n");
    gets(filename);

    if ((fp = fopen(filename, "r")) == 0)
    {
        printf("Can't open file %s. Bye.\n", filename);
        exit(EXIT_FAILURE);
    }
    InitializeTree(&wordcount);

    while (fscanf(fp, "%s", word) == 1 && !TreeIsFull(&wordcount))
    {
        strcpy(entry.wrd, word);
        AddItem(&entry, &wordcount);
    }

    while ((choice = menu()) != 'q')
    {
        switch (choice)
        {
            case 's' :  showwords(&wordcount);
                        break;
            case 'f' :  findword(&wordcount);
                        break;
            default  :  puts("Switching error");
        }
    }
```

```
   fclose(fp);
   puts("Done");
   return 0;
}

char menu(void)
{
    int ch;

    puts("Word counting program");
    puts("Enter the letter corresponding to your choice:");
    puts("s) show word list     f) find a word");
    puts("q) quit");
    while ((ch = getchar()) != EOF)
    {
        while (getchar() != '\n')  /* discard rest of line */
            continue;
        ch = tolower(ch);
        if (strchr("sfq",ch) == NULL)
            puts("Please enter an s, f, or q:");
        else
            break;
    }
    if (ch == EOF)       /* make EOF cause program to quit */
        ch = 'q';

    return ch;
}

void showwords (const Tree * pt)
{
    if (TreeIsEmpty(pt))
        puts("No entries!");
    else
        Traverse(pt, printitem);
}

void findword (const Tree * pt)
{
    char word[SLEN];
    Item entry;
    const Item * pi;

    if (TreeIsEmpty(pt))
    {
        puts("No entries!");
        return;     /* quit function if tree is empty */
    }

    printf("Enter the word to find: ");
    scanf("%s", word);
    while (getchar() != '\n')
        continue;
    strcpy(entry.wrd, word);
    pi = WhereInTree(&entry, pt);
    if (pi == NULL)
```

```
        printf("%s is not in the list.\n", word);
    else
        printf("%s appears %d times.\n", word, pi->count);
}

void printitem(Item item)
{
    printf("%3d:  %s\n", item.count,
            item.wrd);
}


/* pe17-7b.c -- copy of tree.c -- tree support functions */
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "pe17-7.h"

/* local data type */
typedef struct pair {
    Node * parent;
    Node * child;
} Pair;

/* protototypes for local functions */
static Node * MakeNode(const Item * pi);
static bool ToLeft(const Item * i1, const Item * i2);
static bool ToRight(const Item * i1, const Item * i2);
static void AddNode (Node * new_node, Node * root);
static void InOrder(const Node * root, void (* pfun)(Item item));
static Pair SeekItem(const Item * pi, const Tree * ptree);
static void DeleteNode(Node **ptr);
static void DeleteAllNodes(Node * ptr);

/* function definitions */
void InitializeTree(Tree * ptree)
{
    ptree->root = NULL;
    ptree->size = 0;
}

bool TreeIsEmpty(const Tree * ptree)
{
    if (ptree->root == NULL)
        return true;
    else
        return false;
}

bool TreeIsFull(const Tree * ptree)
{
    if (ptree->size == MAXITEMS)
        return true;
    else
        return false;
}
```

```
int TreeItemCount(const Tree * ptree)
{
    return ptree->size;
}

bool AddItem(const Item * pi, Tree * ptree)
{
    Node * new;
    Pair  seek;

    if (TreeIsFull(ptree))
    {
        fprintf(stderr,"Tree is full\n");
        return false;             /* early return            */
    }
    if ((seek = SeekItem(pi, ptree)).child != NULL)
    {
        seek.child->item.count++;
        return true;             /* early return            */
    }
    new = MakeNode(pi);           /* new points to new node */
    if (new == NULL)
    {
        fprintf(stderr, "Couldn't create node\n");
        return false;             /* early return            */
    }
    /* succeeded in creating a new node */
    ptree->size++;

    if (ptree->root == NULL)      /* case 1: tree is empty  */
        ptree->root = new;        /* new node is tree root  */
    else                          /* case 2: not empty      */
        AddNode(new,ptree->root); /* add new node to tree   */
    return true;
}

bool InTree(const Item * pi, const Tree * ptree)
{
    return (SeekItem(pi, ptree).child == NULL) ? false : true;
}

const Item * WhereInTree(const Item * pi, const Tree * ptree)
{
    Node * pn;
    pn = SeekItem(pi,ptree).child;
    if (pn != NULL)
        return &(pn->item);
    else return NULL;
}

bool DeleteItem(const Item * pi, Tree * ptree)
{
    Pair look;
    look = SeekItem(pi, ptree);
    if (look.child == NULL)
        return false;
    if (look.child->item.count > 0)
            look.child->item.count--;
    else
    {
        if (look.parent == NULL)      /* delete root item       */
            DeleteNode(&ptree->root);
        else if (look.parent->left == look.child)
            DeleteNode(&look.parent->left);
        else
            DeleteNode(&look.parent->right);
        ptree->size--;
    }
    return true;
}

void Traverse (const Tree * ptree, void (* pfun)(Item item))
{

    if (ptree != NULL)
        InOrder(ptree->root, pfun);
}

void DeleteAll(Tree * ptree)
{
    if (ptree != NULL)
        DeleteAllNodes(ptree->root);
    ptree->root = NULL;
    ptree->size = 0;
}


/* local functions */
static void InOrder(const Node * root, void (* pfun)(Item item))
{
    if (root != NULL)
    {
        InOrder(root->left, pfun);
        (*pfun)(root->item);
        InOrder(root->right, pfun);
    }
}

static void DeleteAllNodes(Node * root)
{
    Node * pright;

    if (root != NULL)
    {
        pright = root->right;
        DeleteAllNodes(root->left);
        free(root);
        DeleteAllNodes(pright);
    }
}

static void AddNode (Node * new_node, Node * root)
{
    if (ToLeft(&new_node->item, &root->item))
    {
        if (root->left == NULL)      /* empty subtree       */
            root->left = new_node;   /* so add node here    */
        else
            AddNode(new_node, root->left);/* else process subtree*/
    }
    else if (ToRight(&new_node->item, &root->item))
    {
        if (root->right == NULL)
            root->right = new_node;
        else
            AddNode(new_node, root->right);
    }
    else                         /* should be no duplicates */
    {
        fprintf(stderr,"location error in AddNode()\n");
        exit(1);
    }
}

static bool ToLeft(const Item * i1, const Item * i2)
{
    if (strcmp(i1->wrd, i2->wrd) < 0)
        return true;
    else
        return false;
}

static bool ToRight(const Item * i1, const Item * i2)
{
    if (strcmp(i1->wrd, i2->wrd) > 0)
        return true;
    else
        return false;
}

static Node * MakeNode(const Item * pi)
{
    Node * new_node;

    new_node = (Node *) malloc(sizeof(Node));
    if (new_node != NULL)
    {
        new_node->item = *pi;
        new_node->item.count = 1;
        new_node->left = NULL;
        new_node->right = NULL;
    }
    return new_node;
}

static Pair SeekItem(const Item * pi, const Tree * ptree)
{
    Pair look;
    look.parent = NULL;
    look.child = ptree->root;
    if (look.child == NULL)
        return look;                        /* early return   */

    while (look.child != NULL)
    {
        if (ToLeft(pi, &(look.child->item)))
        {
            look.parent = look.child;
            look.child = look.child->left;
        }
        else if (ToRight(pi, &(look.child->item)))
        {
            look.parent = look.child;
            look.child = look.child->right;
        }
        else       /* must be same if not to left or right    */
            break; /* look.child is address of node with item */
    }

    return look;                       /* successful return   */
}

static void DeleteNode(Node **ptr)
/* ptr is address of parent member pointing to target node  */
{
    Node * temp;

    if ( (*ptr)->left == NULL)
    {
        temp = *ptr;
        *ptr = (*ptr)->right;
        free(temp);
    }
    else if ( (*ptr)->right == NULL)
    {
        temp = *ptr;
        *ptr = (*ptr)->left;
        free(temp);
    }
    else    /* deleted node has two children */
    {
        /* find where to reattach right subtree */
        for (temp = (*ptr)->left; temp->right != NULL;
                temp = temp->right)
            continue;
        temp->right = (*ptr)->right;
        temp = *ptr;
        *ptr =(*ptr)->left;
        free(temp);
    }
}
```