C PROGRAMMING *A Modern Approach*

Answers to Even-Numbered Exercises

Susan A. Cole
K. N. King

W · W · Norton & Company
New York · London

W. W. Norton & Company, Inc.
500 Fifth Avenue, New York, N.Y. 10110
http://www.wwnorton.com

W. W. Norton & Company Ltd.
10 Coptic Street, London WC1A 1PU

Last updated: August 19, 1996
Send corrections to *knking@gsu.edu*

2 C Fundamentals

2. (a) The program contains one directive (`#include`) and four statements (three calls of `printf` and one `return`).

(b)
```
Parkinson's Law:
Work expands so as to fill the time
available for its completion.
```

4.
```
#include <stdio.h>

main()
{
  int height = 8, length = 12, width = 10, volume;

  volume = height * length * width;

  printf("Dimensions: %dx%dx%d\n", length, width, height);
  printf("Volume (cubic inches): %d\n", volume);
  printf("Dimensional weight (pounds): %d\n", (volume + 165) / 166);

  return 0;
}
```

6. Here's one possible program:

```
#include <stdio.h>

main()
{
  int i, j, k;
  float x, y, z;

  printf("Value of i: %d\n", i);
  printf("Value of j: %d\n", j);
  printf("Value of k: %d\n", k);

  printf("Value of x: %g\n", x);
  printf("Value of y: %g\n", y);
  printf("Value of z: %g\n", z);

  return 0;
}
```

When compiled using Borland C++ 5.0 and then executed, this program produced the following output:

```
Value of i: 0
Value of j: 0
Value of k: 4223020
Value of x: 1.4013e-45
Value of y: 5.92554e-39
Value of z: 5.90822e-39
```

The values printed depend on many factors, so the chance that you'll get exactly these numbers is small.

8.

```
#include <stdio.h>

main()
{
  float original_amount, amount_with_tax;

  printf("Enter a dollar amount: ");
  scanf("%f", &original_amount);
  amount_with_tax = original_amount * 1.05;
  printf("With 5%% tax added: %.2f\n", amount_with_tax);

  return 0;
}
```

The `amount_with_tax` variable is unnecessary. If we remove it, the program is slightly shorter:

```
#include <stdio.h>

main()
{
  float original_amount;

  printf("Enter a dollar amount: ");
  scanf("%f", &original_amount);
  printf("With 5% tax added: %.2f\n", original_amount * 1.05);

  return 0;
}
```

10. (a) is not legal because `100_bottles` begins with a digit.

12. There are 14 tokens: `a`, `=`, `(`, `3`, `*`, `q`, `-`, `p`, `*`, `p`, `)`, `/`, `3`, and `;`.

3 Formatted Input/Output

2. (a) `printf("%-8.1e", x);`
(b) `printf("%10.6e", x);`
(c) `printf("%-8.3f", x);`
(d) `printf("%6.0f", x);`

4.
```
#include <stdio.h>

main()
{
  int month, day, year;

  printf("Enter a date (mm/dd/yy): ");
  scanf("%d/%d/%d", &month, &day, &year);
  printf("You entered the date %.2d%.2d%.2d\n", year, month, day);

  return 0;
}
```

6.
```
#include <stdio.h>

main()
{
  int language, publisher, book_number, check_digit;

  printf("Enter ISBN: ");
  scanf("%d-%d-%d-%d", &language, &publisher, &book_number, &check_digit);

  printf("Language: %d\n", language);
  printf("Publisher: %d\n", publisher);
  printf("Book number: %d\n", book_number);
  printf("Check digit: %d\n", check_digit);

  /* The four printf calls can be combined as follows:
     printf("Language: %d\nPublisher: %d\nBook number: %d\nCheck digit: %d\n",
            language, publisher, book_number, check_digit);
  */

  return 0;
}
```

8. The values of `x`, `i`, and `y` will be 12.3, 45, and .6, respectively.

4 Expressions

2. No. Suppose that `i` is 9 and `j` is 7. The value of `(-i)/j` could be either –1 or –2, depending on the implementation (see the bottom of page 46). On the other hand, the value of `-(i/j)` is always –1, regardless of the implementation.

4.
```
#include <stdio.h>

main()
{
  int n;

  printf("Enter a three-digit number: ");
  scanf("%d", &n);
  printf("The reversal is: %d%d%d\n", n % 10, (n / 10) % 10, n / 100);

  return 0;
}
```

6. (a) `63 8`
(b) `3 2 1`
(c) `2 -1 3`
(d) `0 0 0`

8. The expression `++i` is equivalent to `(i += 1)`. The value of both expressions is `i` *after* the increment has been performed.

10. There are three possible values for this expression. Assuming that `i` has the value 1 initially, here are the possibilities: (a) Both the increment and the decrement take place after the addition. The value of the expression is 1 + 1 = 2. (b) `i++` is evaluated first (yielding 1), immediately followed by the increment of `i`. The value of `i--` will be 2, so the expression will have the value 1 + 2 = 3. (c) `i--` is evaluated first (yielding 1), immediately followed by the decrement of `i`. The value of `i++` will be 0, so the expression will have the value 0 + 1 = 1.

5 Selection Statements

2. (a) `1`
(b) `1`
(c) `1`
(d) `1`

4.
```
(i > j) - (i < j)
```

6.
```
#include <stdio.h>

main()
{
  int hours, minutes;

  printf("Enter a 24-hour time: ");
  scanf("%d:%d", &hours, &minutes);
    /* assumption: midnight entered as 00:00, valid times are 00:00-23:59 */

  printf("Equivalent 12-hour time: ");
  if (hours == 0)
    printf("12:%.2d AM\n", minutes);
  else if (hours < 12)
    printf("%d:%.2d AM\n", hours, minutes);
  else if (hours == 12)
    printf("%d:%.2d PM\n", hours, minutes);
  else
    printf("%d:%.2d PM\n", hours % 12, minutes);

  return 0;
}
```

8.
```
#include <stdio.h>

main()
{
  int velocity;

  printf("Enter a wind velocity in knots: ");
  scanf("%d", &velocity);

  if (velocity < 1)
    printf("Calm\n");
  else if (velocity <= 3)
    printf("Light air\n");
  else if (velocity <= 27)
    printf("Breeze\n");
  else if (velocity <= 47)
    printf("Gale\n");
  else if (velocity <= 63)
    printf("Storm\n");
  else
    printf("Hurricane\n");

  return 0;
}
```

10.
```
#include <stdio.h>

main ()
{
  int check_digit, d, i1, i2, i3, i4, i5, j1, j2, j3, j4, j5,
      first_sum, second_sum, total;

  printf("Enter the first (single) digit: ");
  scanf("%1d", &d);
  printf("Enter the first group of five digits: ");
  scanf("%1d%1d%1d%1d%1d", &i1, &i2, &i3, &i4, &i5);
  printf("Enter the second group of five digits: ");
  scanf("%1d%1d%1d%1d%1d", &j1, &j2, &j3, &j4, &j5);
  printf("Enter the last (single) digit: ");
  scanf("%1d", &check_digit);

  first_sum = d + i2 + i4 + j1 + j3 + j5;
  second_sum = i1 + i3 + i5 + j2 + j4;
  total = 3 * first_sum + second_sum;

  if (check_digit == 9-((total-1)%10))
    printf("VALID\n");
  else
    printf("NOT VALID\n");

  return 0;
}
```

12. Yes, the statement is legal. When n is equal to 5, it does nothing, since 5 is not equal to –9.

14.
```
#include <stdio.h>

main()
{
  int grade;

  printf("Enter numerical grade: ");
  scanf("%d", &grade);

  if (grade < 0 || grade > 100) {
    printf("Illegal grade\n");
    return 0;
  }

  switch (grade / 10) {
    case 10:
    case 9:  printf("Letter grade: A\n");
             break;
    case 8:  printf("Letter grade: B\n");
             break;
    case 7:  printf("Letter grade: C\n");
             break;
    case 6:  printf("Letter grade: D\n");
             break;
    case 5:
    case 4:
    case 3:
    case 2:
    case 1:
    case 0:  printf("Letter grade: F\n");
             break;
  }

  return 0;
}
```

16. The output is

```
onetwo
```

since there are no `break` statements after the cases.

6 Loops

2.

```
#include <stdio.h>

main()
{
  int m, n, remainder;

  printf("Enter two integers: ");
  scanf("%d%d", &m, &n);

  while (n != 0) {
    remainder = m % n;
    m = n;
    n = remainder;
  }

  printf("Greatest common divisor: %d\n", m);

  return 0;
}
```

4.

```
#include <stdio.h>

main()
{
  float commission, value;

  printf("Enter value of trade: ");
  scanf("%f", &value);

  while (value != 0.0) {
    if (value < 2500.00)
      commission = 30.00 + .017 * value;
    else if (value < 6250.00)
      commission = 56.00 + .0066 * value;
    else if (value < 20000.00)
      commission = 76.00 + .0034 * value;
    else if (value < 50000.00)
      commission = 100.00 + .0022 * value;
    else if (value < 500000.00)
      commission = 155.00 + .0011 * value;
    else
      commission = 255.00 + .0009 * value;

    if (commission < 39.00)
      commission = 39.00;

    printf("Commission: $%.2f\n\n", commission);

    printf("Enter value of trade: ");
    scanf("%f", &value);
  }

  return 0;
}
```

6.
```
#include <stdio.h>

main()
{
  int i, n;

  printf("Enter limit on maximum square: ");
  scanf("%d", &n);

  for (i = 2; i * i <= n; i += 2)
    printf("%d\n", i * i);

  return 0;
}
```

8.
```
#include <stdio.h>

main()
{
  int i, n, start_day;

  printf("Enter number of days in month: ");
  scanf("%d", &n);
  printf("Enter starting day of the week (1=Sun, 7=Sat): ");
  scanf("%d", &start_day);

  /* print any leading "blank dates" */
  for (i = 1; i < start_day; i++)
    printf("   ");

  /* now print the calendar */
  for (i = 1; i <= n; i++) {
    printf("%3d", i);
    if ((start_day + i - 1) % 7 == 0)
      printf("\n");
  }

  return 0;
}
```

10. (c) is not equivalent to (a) and (b), because `i` is incremented *before* the loop body is executed.

12. Consider the following `while` loop:

```
while (…) {
  …
  continue;
  …
}
```

The equivalent code using `goto` would have the following appearance:

```
while (…) {
  …
  goto loop_end;
  …
  loop_end: ;   /* null statement */
}
```

14.
```
for (d = 2; d * d <= n; d++)
  if (n % d == 0) break;
```

The `if` statement that follows the loop will need to be modified as well:

```
if (d * d <= n)
  printf("%d is divisible by %d\n", n, d);
else
  printf("%d is prime\n", n);
```

16. The problem is the semicolon at the end of the first line. If we remove it, the statement is now correct:

```
if (n % 2 == 0)
  printf("n is even\n");
```

7 Basic Types

2. If `int` values are stored in 16 bits, entering 182 will cause the program to print a nonsense value for the last square. If `int` values are stored in 32 bits, entering 46341 causes failure. `short int` values are usually stored in 16 bits, causing failure at 182. `long int` values are usually stored in 32 bits, causing failure at 46341.

4. (b) is not legal.

6. (d) is illegal, since `printf` requires a string, not a character, as its first argument.

8.
```
#include <stdio.h>

main()
{
  int i, n;
  char ch;

  printf("This program prints a table of squares.\n");
  printf("Enter number of entries in table: ");
  scanf("%d", &n);
  ch = getchar(); /* dispose of new-line character following number of entries */

  for (i = 1; i <= n; i++) {
    printf("%10d%10d\n", i, i * i);
    if (i % 24 == 0) {
      printf("Press Enter to continue...");
      ch = getchar();   /* or simply getchar(); */
    }
  }

  return 0;
}
```

10.
```
#include <ctype.h>
#include <stdio.h>

main()
{
  int sum = 0;
  char ch;

  printf("Enter a word: ");

  while ((ch = getchar()) != '\n')
    switch (toupper(ch)) {
      case 'D': case 'G':
        sum += 2; break;
      case 'B': case 'C': case 'M': case 'P':
        sum += 3; break;
      case 'F': case 'H': case 'V': case 'W': case 'Y':
        sum += 4; break;
      case 'K':
        sum += 5; break;
      case 'J': case 'X':
        sum += 8; break;
```

```
      case 'Q': case 'Z':
        sum += 10; break;
      default:
        sum++; break;
    }

  printf("Scrabble value: %d\n", sum);

  return 0;
}
```

12.

```
#include <stdio.h>

main()
{
  printf("Size of int: %d\n", (int) sizeof(int));
  printf("Size of short int: %d\n", (int) sizeof(short int));
  printf("Size of long int: %d\n", (int) sizeof(long int));
  printf("Size of float: %d\n", (int) sizeof(float));
  printf("Size of double: %d\n", (int) sizeof(double));
  printf("Size of long double: %d\n", (int) sizeof(long double));

  return 0;
}
```

Since the type of a `sizeof` expression may vary from one implementation to another, it's necessary to cast `sizeof` expressions to a known type before printing them. The sizes of the basic types are small numbers, so it's safe to cast them to `int` type. In general, however, it's best to cast `sizeof` expressions to `unsigned long int` and print them using `%lu`.

14. `unsigned int`, because the `(int)` cast applies only to `j`, not `j * k`.

16. The value of `i` is converted to `float` and added to `f`, then the result is converted to `double` and stored in `d`.

18. No. Converting `f` to `int` will fail if the value stored in `f` exceeds the largest value of type `int`.

2.

```
#include <stdio.h>

main()
{
  int digit_count[10] = {0};
  int digit;
  long int n;

  printf("Enter a number: ");
  scanf("%ld", &n);

  while (n > 0) {
    digit = n % 10;
    digit_count[digit]++;
    n /= 10;
  }

  printf ("Digit:      ");
  for (digit = 0; digit <= 9; digit++)
    printf("%3d", digit);
  printf("\nOccurrences:");
  for (digit = 0; digit <= 9; digit++)
    printf("%3d", digit_count[digit]);
  printf("\n");

  return 0;
}
```

4. The problem with `sizeof(a) / sizeof(`*t*`)` is that it can't easily be checked for correctness by someone reading the program. (The reader would have to locate the declaration of `a` and make sure that its elements have type *t*.)

6.

```
#include <stdio.h>

#define NUM_RATES (sizeof(value)/sizeof(value[0]))
#define INITIAL_BALANCE 100.00

main()
{
  int i, low_rate, month, num_years, year;
  float value[5];

  printf("Enter interest rate: ");
  scanf("%d", &low_rate);
  printf("Enter number of years: ");
  scanf("%d", &num_years);

  printf("\nYears");
  for (i = 0; i < NUM_RATES; i++) {
    printf("%6d%%", low_rate+i);
    value[i] = INITIAL_BALANCE;
  }
  printf("\n");
```

```
  for (year = 1; year <= num_years; year++) {
    printf("%3d    ", year);
    for (i = 0; i < NUM_RATES; i++) {
      for (month = 1; month <= 12; month++)
        value[i] += ((float)(low_rate+i)/12) / 100.0 * value[i];
      printf("%7.2f", value[i]);
    }
    printf("\n");
  }

  return 0;
}
```

8. To use a digit `d` (in character form) as a subscript into the array `a`, we would write `a[d-'0']`. We're assuming that digits have consecutive codes in the underlying character set, which is true of ASCII and other popular character sets.

10.
```
const int segments[10][7] = {{1, 1, 1, 1, 1, 1},
                             {0, 1, 1},
                             {1, 1, 0, 1, 1, 0, 1},
                             {1, 1, 1, 1, 0, 0, 1},
                             {0, 1, 1, 0, 0, 1, 1},
                             {1, 0, 1, 1, 0, 1, 1},
                             {1, 0, 1, 1, 1, 1, 1},
                             {1, 1, 1},
                             {1, 1, 1, 1, 1, 1, 1},
                             {1, 1, 1, 1, 0, 1, 1}};
```

12.
```
#include <stdio.h>

#define NUM_QUIZZES  5
#define NUM_STUDENTS 5

main()
{
  int grades[NUM_STUDENTS][NUM_QUIZZES];
  int high, low, quiz, student, total;

  for (student = 0; student < NUM_STUDENTS; student++) {
    printf("Enter grades for student %d: ", student + 1);
    for (quiz = 0; quiz < NUM_QUIZZES; quiz++)
      scanf("%d", &grades[student][quiz]);
  }

  printf("\nStudent  Total  Average\n");
  for (student = 0; student < NUM_STUDENTS; student++) {
    printf("%4d      ", student + 1);
    total = 0;
    for (quiz = 0; quiz < NUM_QUIZZES; quiz++)
      total += grades[student][quiz];
    printf("%3d     %3d\n", total, total / NUM_QUIZZES);
  }

  printf("\nQuiz  Average  High  Low\n");
  for (quiz = 0; quiz < NUM_QUIZZES; quiz++) {
    printf("%3d     ", quiz + 1);
    total = 0;
    high = 0;
    low = 100;
```

```
    for (student = 0; student < NUM_STUDENTS; student++) {
      total += grades[student][quiz];
      if (grades[student][quiz] > high)
        high = grades[student][quiz];
      if (grades[student][quiz] < low)
        low = grades[student][quiz];
    }
    printf("%3d    %3d   %3d\n", total / NUM_STUDENTS, high, low);
  }

  return 0;
}
```

9 Functions

2.
```
int check(int x, int y, int n)   /* assumes n is positive */
{
  return (x >= 0 && x <= n - 1 && y >= 0 && y <= n - 1);
}
```

4.
```
int day_of_year(int month, int day, int year)
{
  int num_days[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
  int day_count = 0, i;

  for (i = 1; i < month; i++)
    day_count += num_days[i-1];

  /* adjust for leap years, assuming that leap years are divisible by 4 */
  if (year % 4 == 0 && month > 2)
    day_count++;

  return day_count + day;
}
```

Using the expression `year % 4 == 0` to test for leap years is not completely correct. The correct test is

```
year % 4 == 0 && (year % 100 != 0 || year % 400 == 0)
```

6.
```
int digit(int n, int k)
{
  int i;

  for (i = 1; i < k; i++)
    n /= 10;

  return n != 0 ? n % 10 : -1;
}
```

8. (a) and (b) are valid prototypes. (c) is illegal, since it doesn't specify the type of the argument. (d) incorrectly specifies that `f` returns an `int` value.

10. (a)
```
int largest(int a[], int n)
{
  int i, max = a[0];

  for (i = 1; i < n; i++)
    if (a[i] > max)
      max = a[i];

  return max;
}
```

(b)
```
int average(int a[], int n)
{
  int i, avg = 0;

  for (i = 0; i < n; i++)
    avg += a[i];

  return avg / n;
}
```

(c)
```
int num_positive(int a[], int n)
{
  int i, count = 0;

  for (i = 0; i < n; i++)
    if (a[i] > 0)
      count++;

  return count;
}
```

12.
```
float median(float x, float y, float z)
{
  float result;

  if (x <= y)
    if (y <= z) result = y;
    else if (x <= z) result = z;
    else result = x;
  else {
    if (z <= y) result = y;
    else if (x <= z) result = x;
    else result = z;
  }

  return result;
}
```

14.
```
int fact(int n)
{
  int i, result = 1;

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

  return result;
}
```

16. The following program tests the `pb` function:

```
#include <stdio.h>

void pb(int n);

main()
{
  int n;

  printf("Enter a number: ");
  scanf("%d", &n);

  printf("Output of pb: ");
  pb(n);
  printf("\n");

  return 0;
}
```

```
void pb(int n)
{
  if (n != 0) {
    pb(n / 2);
    putchar('0' + n % 2);
  }
}
```

pb prints the binary representation of the argument n, assuming that n is greater than 0. (We also assume that digits have consecutive codes in the underlying character set.) For example:

```
Enter a number: 53
Output of pb: 110101
```

A trace of pb's execution would look like this:

pb(53) finds that 53 is not equal to 0, so it calls
pb(26), which finds that 26 is not equal to 0, so it calls
pb(13), which finds that 13 is not equal to 0, so it calls
pb(6), which finds that 6 is not equal to 0, so it calls
pb(3), which finds that 3 is not equal to 0, so it calls
pb(1), which finds that 1 is not equal to 0, so it calls
pb(0), which finds that 0 *is* equal to 0, so it returns, causing
pb(1) to print 1 and return, causing
pb(3) to print 1 and return, causing
pb(6) to print 0 and return, causing
pb(13) to print 1 and return, causing
pb(26) to print 0 and return, causing
pb(53) to print 1 and return.

10 Program Organization

2. (a) a, b, and c are visible.
(b) a, and d are visible.
(c) a, d, and e are visible.
(d) a and f are visible.

4.
```
#include <stdio.h>
#include <stdlib.h>

#define NUM_CARDS 5
#define RANK 0
#define SUIT 1
#define TRUE 1
#define FALSE 0

typedef int Bool;

int hand[NUM_CARDS][2];
/*    0    1
     ____ ____
 0 |____|____|
 1 |____|____|
 2 |____|____|
 3 |____|____|
 4 |____|____|
    rank suit
*/

Bool straight, flush, four, three;
int pairs;   /* can be 0, 1, or 2 */

void read_cards(void);
void analyze_hand(void);
void print_result(void);

/**********************************************************
 * main: Calls read_cards, analyze_hand, and print_result *
 *       repeatedly.                                      *
 **********************************************************/
main()
{
  for (;;) {          /* infinite loop */
    read_cards();
    analyze_hand();
    print_result();
  }
}

/**********************************************************
 * read_cards: Reads the cards into the external variable *
 *             hand; checks for bad cards and duplicate   *
 *             cards.                                     *
 **********************************************************/
void read_cards(void)
{
  char ch, rank_ch, suit_ch;
  int i, rank, suit;
```

```
    Bool bad_card, duplicate_card;
    int cards_read = 0;

    while (cards_read < NUM_CARDS) {

      bad_card = FALSE;

      printf("Enter a card: ");

      rank_ch = getchar();
      switch (rank_ch) {
        case '0':           exit(EXIT_SUCCESS);
        case '2':           rank = 0; break;
        case '3':           rank = 1; break;
        case '4':           rank = 2; break;
        case '5':           rank = 3; break;
        case '6':           rank = 4; break;
        case '7':           rank = 5; break;
        case '8':           rank = 6; break;
        case '9':           rank = 7; break;
        case 't': case 'T': rank = 8; break;
        case 'j': case 'J': rank = 9; break;
        case 'q': case 'Q': rank = 10; break;
        case 'k': case 'K': rank = 11; break;
        case 'a': case 'A': rank = 12; break;
        default:            bad_card = TRUE;
      }

      suit_ch = getchar();
      switch (suit_ch) {
        case 'c': case 'C': suit = 0; break;
        case 'd': case 'D': suit = 1; break;
        case 'h': case 'H': suit = 2; break;
        case 's': case 'S': suit = 3; break;
        default:            bad_card = TRUE;
      }

      while ((ch = getchar()) != '\n')
        if (ch != ' ') bad_card = TRUE;

      if (bad_card) {
        printf("Bad card; ignored.\n");
        continue;
      }

      duplicate_card = FALSE;
      for (i = 0; i < cards_read; i++)
        if (hand[i][RANK] == rank && hand[i][SUIT] == suit) {
          printf("Duplicate card; ignored.\n");
          duplicate_card = TRUE;
          break;
        }

      if (!duplicate_card) {
        hand[cards_read][RANK] = rank;
        hand[cards_read][SUIT] = suit;
        cards_read++;
      }
    }
  }
```

```
/*****************************************************
 * analyze_hand: Determines whether the hand contains a   *
 *               straight, a flush, four-of-a-kind,       *
 *               and/or a three-of-a-kind; determines the *
 *               number of pairs; stores the results into *
 *               the external variables straight, flush,  *
 *               four, three, and pairs.                  *
 *****************************************************/
void analyze_hand(void)
{
  int rank, suit, card, pass, run;

  straight = TRUE;
  flush = TRUE;
  four = FALSE;
  three = FALSE;
  pairs = 0;

  /* sort cards by rank */
  for (pass = 1; pass < NUM_CARDS; pass++)
    for (card = 0; card < NUM_CARDS-pass; card++) {
      rank = hand[card][RANK];
      suit = hand[card][SUIT];
      if (hand[card+1][RANK] < rank) {
        hand[card][RANK] = hand[card+1][RANK];
        hand[card][SUIT] = hand[card+1][SUIT];
        hand[card+1][RANK] = rank;
        hand[card+1][SUIT] = suit;
      }
    }

  /* check for flush */
  suit = hand[0][SUIT];
  for (card = 1; card < NUM_CARDS; card++)
    if (hand[card][SUIT] != suit)
      flush = FALSE;


  /* check for straight */
  for (card = 0; card < NUM_CARDS-1; card++)
    if (hand[card][RANK] + 1 != hand[card+1][RANK])
      straight = FALSE;

  /* check for 4-of-a-kind, 3-of-a-kind, and pairs by
     looking for "runs" of cards with identical ranks */
  card = 0;
  while (card < NUM_CARDS) {
    rank = hand[card][RANK];
    run = 0;
    do {
      run++;
      card++;
    } while (card < NUM_CARDS && hand[card][RANK] == rank);
    switch (run) {
      case 2: pairs++;      break;
      case 3: three = TRUE; break;
      case 4: four = TRUE;  break;
    }
  }
}
```

```
/*********************************************************
 * print_result: Notifies the user of the result, using   *
 *               the external variables straight, flush,  *
 *               four, three, and pairs.                  *
 *********************************************************/
void print_result(void)
{
  if (straight && flush) printf("Straight flush\n\n");
  else if (four)         printf("Four of a kind\n\n");
  else if (three &&
           pairs == 1)   printf("Full house\n\n");
  else if (flush)        printf("Flush\n\n");
  else if (straight)     printf("Straight\n\n");
  else if (three)        printf("Three of a kind\n\n");
  else if (pairs == 2)   printf("Two pairs\n\n");
  else if (pairs == 1)   printf("Pair\n\n");
  else                   printf("High card\n\n");
}
```

6.

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

#define NUM_RANKS 13
#define NUM_SUITS 4
#define NUM_CARDS 5
#define TRUE 1
#define FALSE 0

typedef int Bool;

int num_in_rank[NUM_RANKS+1];
  /* NOTE: Aces are stored twice, in first and last
           elements of num_in_rank */
int num_in_suit[NUM_SUITS];
Bool straight, flush, four, three;
int pairs;   /* can be 0, 1, or 2 */

void read_cards(void);
void analyze_hand(void);
void print_result(void);

/*********************************************************
 * main: Calls read_cards, analyze_hand, and print_result *
 *       repeatedly.                                      *
 *********************************************************/
main()
{
  for (;;) {          /* infinite loop */
    read_cards();
    analyze_hand();
    print_result();
  }
}

/*********************************************************
 * read_cards: Reads the cards into the external          *
 *             variables num_in_rank and num_in_suit;     *
 *             checks for bad cards and duplicate cards.  *
 *********************************************************/
void read_cards(void)
{
  Bool card_exists[NUM_RANKS][NUM_SUITS];
  char ch, rank_ch, suit_ch;
```

```
    int rank, suit;
    Bool bad_card;
    int cards_read = 0;

    for (rank = 0; rank < NUM_RANKS; rank++) {
      num_in_rank[rank] = 0;
      for (suit = 0; suit < NUM_SUITS; suit++)
        card_exists[rank][suit] = FALSE;
    }

    for (suit = 0; suit < NUM_SUITS; suit++)
      num_in_suit[suit] = 0;

    while (cards_read < NUM_CARDS) {

      bad_card = FALSE;

      printf("Enter a card: ");

      rank_ch = getchar();
      switch (rank_ch) {
        case '0':           exit(EXIT_SUCCESS);
        case '2':           rank = 1; break;
        case '3':           rank = 2; break;
        case '4':           rank = 3; break;
        case '5':           rank = 4; break;
        case '6':           rank = 5; break;
        case '7':           rank = 6; break;
        case '8':           rank = 7; break;
        case '9':           rank = 8; break;
        case 't': case 'T': rank = 9; break;
        case 'j': case 'J': rank = 10; break;
        case 'q': case 'Q': rank = 11; break;
        case 'k': case 'K': rank = 12; break;
        case 'a': case 'A': rank = 0; break;
          /* NOTE: Ranks have been renumbered so that aces
                   are low */
        default:            bad_card = TRUE;
      }

      suit_ch = getchar();
      switch (suit_ch) {
        case 'c': case 'C': suit = 0; break;
        case 'd': case 'D': suit = 1; break;
        case 'h': case 'H': suit = 2; break;
        case 's': case 'S': suit = 3; break;
        default:            bad_card = TRUE;
      }

      while ((ch = getchar()) != '\n')
        if (ch != ' ') bad_card = TRUE;

      if (bad_card)
        printf("Bad card; ignored.\n");
      else if (card_exists[rank][suit])
        printf("Duplicate card; ignored.\n");
      else {
        num_in_rank[rank]++;
        num_in_suit[suit]++;
        card_exists[rank][suit] = TRUE;
        cards_read++;
      }
    }
  }
```

```
/*************************************************************
 * analyze_hand: Determines whether the hand contains a      *
 *               straight, a flush, four-of-a-kind,          *
 *               and/or a three-of-a-kind; determines the    *
 *               number of pairs; stores the results into    *
 *               the external variables straight, flush,     *
 *               four, three, and pairs.                     *
 *************************************************************/
void analyze_hand(void)
{
  int num_consec = 0;
  int rank, suit;

  straight = FALSE;
  flush = FALSE;
  four = FALSE;
  three = FALSE;
  pairs = 0;

  /* check for flush */
  for (suit = 0; suit < NUM_SUITS; suit++)
    if (num_in_suit[suit] == NUM_CARDS)
      flush = TRUE;

  /* check for straight; ace-low straights are allowed */
  num_in_rank[NUM_RANKS] = num_in_rank[0];
    /* aces are high as well as low */
  rank = 0;
  while (num_in_rank[rank] == 0) rank++;
  for (; rank < NUM_RANKS+1 && num_in_rank[rank]; rank++)
    num_consec++;
  if (num_consec == NUM_CARDS) {
    straight = TRUE;
    return;
  }

  /* check for 4-of-a-kind, 3-of-a-kind, and pairs */
  for (rank = 0; rank < NUM_RANKS; rank++) {
    if (num_in_rank[rank] == 4) four = TRUE;
    if (num_in_rank[rank] == 3) three = TRUE;
    if (num_in_rank[rank] == 2) pairs++;
  }
}

/*************************************************************
 * print_result: Notifies the user of the result, using      *
 *               the external variables straight, flush,     *
 *               four, three, and pairs.                     *
 *************************************************************/
void print_result(void)
{
  if (straight && flush) printf("Straight flush\n\n");
  else if (four)         printf("Four of a kind\n\n");
  else if (three &&
           pairs == 1)   printf("Full house\n\n");
  else if (flush)        printf("Flush\n\n");
  else if (straight)     printf("Straight\n\n");
  else if (three)        printf("Three of a kind\n\n");
  else if (pairs == 2)   printf("Two pairs\n\n");
  else if (pairs == 1)   printf("Pair\n\n");
  else                   printf("High card\n\n");
}
```

11 Pointers

2. (e), (f), and (i) are legal. (a) is illegal because `p` is a pointer to an integer and `i` is an integer. (b) is illegal because `*p` is an integer and `&i` is a pointer to an integer. (c) is illegal because `&p` is a pointer to a pointer to an integer and `q` is a pointer to an integer. (d) is illegal for reasons similar to (c). (g) is illegal because `p` is pointer to an integer and `*q` is an integer. (h) is illegal because `*p` is an integer and `q` is a pointer to an integer.

4.
```
#include <stdio.h>

void swap(int *p, int *q);

main()
{
  int x = 1, y = 2;

  swap(&x, &y);
  printf("x = %d, y = %d\n", x, y);
  return 0;
}

void swap(int *p, int *q)
{
  int temp;

  temp = *p;
  *p = *q;
  *q = temp;
}
```

6.
```
void find_two_largest(int a[], int n, int *largest, int *second_largest)
{
  int i;

  *largest = a[0];
  *second_largest = a[0];

  for (i = 1; i < n; i++)
    if (a[i] > *largest) {
      *second_largest = *largest;
      *largest = a[i];
    } else if (a[i] > *second_largest)
      *second_largest = a[i];
}
```

12 Pointers and Arrays

2. The statement is illegal because pointers cannot be added. Here's a legal statement that has the same effect:

```
middle = low + (high - low) / 2;
```

The value of `(high - low) / 2` is an integer, not a pointer, so it can legally be added to `low`.

4. (a)
```
#include <stdio.h>

#define MSG_LEN 80     /* maximum length of message */

main()
{
  char msg[MSG_LEN];
  int i;

  printf("Enter a message: ");
  for (i = 0; i < MSG_LEN; i++) {
    msg[i] = getchar();
    if (msg[i] == '\n') break;
  }

  printf("Reversal is: ");
  for (i--; i >= 0; i--)
    putchar(msg[i]);
  putchar('\n');

  return 0;
}
```

(b)
```
#include <stdio.h>

#define MSG_LEN 80     /* maximum length of message */

main()
{
  char msg[MSG_LEN], *p;

  printf("Enter a message: ");
  for (p = &msg[0]; p < &msg[MSG_LEN]; p++) {
    *p = getchar();
    if (*p == '\n') break;
  }

  printf("Reversal is: ");
  for (p--; p >= &msg[0]; p--)
    putchar(*p);
  putchar('\n');

  return 0;
}
```

6.
```
int *top_ptr;

void make_empty(void)
{
  top_ptr = &contents[0];
}
```

```
Bool is_empty(void)
{
  return top_ptr == &contents[0];
}

Bool is_full(void)
{
  return top_ptr == &contents[STACK_SIZE];
}
```

8.

```
#include <stdio.h>

#define MSG_LEN 80     /* maximum length of message */

main()
{
  char msg[MSG_LEN], *p;

  printf("Enter a message: ");
  for (p = msg; p < msg + MSG_LEN; p++) {
    *p = getchar();
    if (*p == '\n') break;
  }

  printf("Reversal is: ");
  for (p--; p >= msg; p--)
    putchar(*p);
  putchar('\n');

  return 0;
}
```

10.

```
int sum_array(int a[], int n)
{
  int *p, sum;

  sum = 0;
  for (p = a; p < a + n; p++)
    sum += *p;
  return sum;
}
```

12.

```
#define N 10

float ident[N][N], *p;
int num_zeros = N;

for (p = &ident[0][0]; p <= &ident[N-1][N-1]; p++)
  if (num_zeros == N) {
    *p = 1.0;
    num_zeros = 0;
  } else {
    *p = 0.0;
    num_zeros++;
  }
```

14.

```
int *p;

for (p = temperatures[i]; p < temperatures[i] + 24; p++)
  printf("%d ", *p);
```

13 Strings

2. (a) Illegal; `p` is not a character.
(b) Legal; output is `a`.
(c) Legal; output is `abc`.
(d) Illegal; `*p` is not a pointer.

4. (a)
```
int read_line(char str[], int n)
{
  char ch;
  int i = 0;

  while ((ch = getchar()) != '\n')
    if (i == 0 && isspace(ch))
      ;   /* ignore */
    else if (i < n)
      str[i++] = ch;

  str[i] = '\0';
  return i;
}
```

(b)
```
int read_line(char str[], int n)
{
  char ch;
  int i = 0;

  while (!isspace(ch = getchar()))
    if (i < n)
      str[i++] = ch;

  str[i] = '\0';
  return i;
}
```

(c)
```
int read_line(char str[], int n)
{
  char ch;
  int i = 0;

  do {
    ch = getchar();
    if (i < n)
      str[i++] = ch;
  } while (ch != '\n');

  str[i] = '\0';
  return i;
}
```

(d)
```
int read_line(char str[], int n)
{
  char ch;
  int i;
```

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

  str[i] = '\0';
  return i;
}
```

6.
```
void censor(char s[])
{
  int i;

  for (i = 0; s[i] != '\0'; i++)
    if (s[i] == 'f' && s[i+1] == 'o' && s[i+2] =='o')
      s[i] = s[i+1] = s[i+2] = 'x';
}
```

Note that the short-circuit evaluation of && prevents the `if` statement from testing characters that follow the null character.

8. (a) 3
(b) 0
(c) The length of the longest prefix of the string `s` that consists entirely of characters from the string `t`. Or, equivalently, the position of the first character in `s` that is not also in `t`.

10.
```
tired-or-wired?
```

12. `q` doesn't point anywhere in particular, so the call of `strcpy` copies the string pointed to by `p` into some unknown area of memory. Exercise 2 in Chapter 17 discusses how to write this function correctly.

14.
```
#include <stdio.h>
#include <string.h>

#define WORD_LEN 20

void read_line(char str[], int n);

main()
{
  char smallest_word[WORD_LEN+1],
       largest_word[WORD_LEN+1],
       current_word[WORD_LEN+1];

  printf("Enter word: ");
  read_line(current_word, WORD_LEN);
  strcpy(smallest_word, strcpy(largest_word, current_word));

  while (strlen(current_word) != 4) {
    printf("Enter word: ");
    read_line(current_word, WORD_LEN);
    if (strcmp(current_word, smallest_word) < 0)
      strcpy(smallest_word, current_word);
    if (strcmp(current_word, largest_word) > 0)
      strcpy(largest_word, current_word);
  }

  printf("\nSmallest word: %s\n", smallest_word);
  printf("Largest word: %s\n", largest_word);

  return 0;
}
```

```
void read_line(char str[], int n)
{
  char ch;
  int i = 0;

  while ((ch = getchar()) != '\n')
    if (i < n)
      str[i++] = ch;

  str[i] = '\0';
}
```

16.

```
int count_spaces(const char *s)
{
  int count = 0;

  while (*s)
    if (*s++ == ' ')
      count++;
  return count;
}
```

18.

```
#include <stdio.h>

main(int argc, char *argv[])
{
  int i;

  for (i = argc - 1; i > 0; i--)
    printf("%s ", argv[i]);

  return 0;
}
```

20.

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

#define NUM_PLANETS 9

int string_equal(const char *s, const char *t);

main(int argc, char *argv[])
{
  char *planets[] = {"Mercury", "Venus", "Earth",
                     "Mars", "Jupiter", "Saturn",
                     "Uranus", "Neptune", "Pluto"};
  int i, j;

  for (i = 1; i < argc; i++) {
    for (j = 0; j < NUM_PLANETS; j++)
      if (string_equal(argv[i], planets[j])) {
        printf("%s is planet %d\n", argv[i], j+1);
        break;
      }
    if (j == NUM_PLANETS)
      printf("%s is not a planet\n", argv[i]);
  }

  return 0;
}
```

```
int string_equal(const char *s, const char *t)
{
  int i;

  for (i = 0; toupper(s[i]) == toupper(t[i]); i++)
    if (s[i] == '\0')
      return 1;

  return 0;
}
```

14 The Preprocessor

2. `#define NELEMS(a) (sizeof(a)/sizeof(a[0]))`

4. (a) One problem stems from the lack of parentheses around the replacement list. For example, the statement

```
a = 1/AVG(b, c);
```

will be replaced by

```
a = 1/(b+c)/2;
```

Even if we add the missing parentheses, though, the macro still has problems, because it needs parentheses around `x` and `y` in the replacement list. Consider the following example:

```
a = AVG(b<c, c>d);
```

becomes

```
a = ((b<c+c>d)/2);
```

which is equivalent to

```
a = ((b<(c+c)>d)/2);
```

Here's the final (corrected) version of the macro:

```
#define AVG(x,y) (((x)+(y))/2)
```

(b) The problem is the lack of parentheses around the replacement list. For example,

```
a = 1/AREA(b, c);
```

becomes

```
a = 1/(b)*(c);
```

Here's the corrected macro:

```
#define AREA(x,y) ((x)*(y))
```

6. (a) The call of `putchar` expands into the following statement:

```
putchar(('a'<=(s[++i])&&(s[++i])<='z'?(s[++i])-'a'+'A':(s[++i])));
```

The character `a` is less than or equal to `s[1]` (which is `b`), yielding a true condition. The character `s[2]` (which is `c`) is less than or equal to `z`, which is also true. The value printed is `s[3]-'a'+'A'`, which is `D`.

(b) The character `a` is not less than or equal to `s[1]` (which is `1`) so the test condition is false. The value printed is `s[2]`, which is `2`.

8. (a)
```
long long_max(long x, long y)
{
  return x > y ? x : y;
}
```

The preprocessor would actually put all the tokens on one line, but this version is more readable.

(b) The problem with types such as `unsigned long` is that they require two words, which prevents `GENERIC_MAX` from creating the desired function name. For example, `GENERIC_MAX(unsigned long)` would expand into

```
unsigned long unsigned long_max(unsigned long x, unsigned long y)
{
  return x > y ? x : y;
}
```

(c) To make `GENERIC_MAX` work with any basic type, use a type definition to rename the type:

```
typedef unsigned long ULONG;
```

We can now write `GENERIC_MAX(ULONG)`.

10. (c) and (e) will fail, since `M` is defined.

12. Here's what the program will look like after preprocessing:

```
Blank line
Blank line
Blank line
Blank line
Blank line
Blank line
Blank line

main()
{
  int a[= 10], i, j, k, m;

Blank line
  i = j;
Blank line
Blank line
Blank line

  i = 10 * j+1;
  i = (x,y) x - y(j, k);
  i = ((((j++)*(j++)))*(((j++)*(j++))));
  i = (((j)*(j))*(j));
  i = jk;
  puts("i" "j");

Blank line
  i = SQR(j);
Blank line
  i = (j);

  return 0;
}
```

Some preprocessors delete white-space characters at the beginning of a line, so your results may vary. Three lines will cause errors when the program is compiled. Two contain syntax errors:

```
int a[= 10], i, j, k, m;
i = (x,y) x - y(j, k);
```

The third refers to an undefined variable:

```
i = jk;
```

15 Writing Large Programs

2. (b). Function definitions should not be put in a header file. If a function definition appears in a header file that is included by two (or more) source files, the program can't be linked, since the linker will see two copies of the function.

4. (a) The program should work correctly if `sizeof(int)` is the same as `sizeof(long int)`.

(b) If `sizeof(int)` is less than `sizeof(long int)`, an assignment to `i` in `bar.c` will modify not only the bytes that belong to `i`, but other bytes that follow it in memory. The effect is unpredictable (it depends on how the compiler allocates memory), but the program is likely to produce incorrect results or even crash.

6. Contents of `stack.h` file:

```
#ifndef STACK_H
#define STACK_H

void make_empty(void);
int is_empty(void);
int is_full(void);
void push(int i);
int pop(void);

#endif
```

Contents of `calc.c` file:

```
#include <ctype.h>
#include <stdio.h>
#include "stack.h"

#define MAX_LEN 20   /* limits length of a token */

char current_char = ' ';   /* retains character between calls of read_token */

int read_token(char *token, int n);
int convert_to_number(const char *token, int *value);
void error(const char *msg);

main()
{
  int op1, op2, result;
  char token[MAX_LEN+1];

  make_empty();
  for (;;) {
    switch (read_token(token, MAX_LEN)) {
      case -1:   /* token too long */
        error("Token too long");
        continue;

      case 0:   /* end of input line */
        if (is_empty())
          return 0;   /* entering an empty line terminates program */
        current_char = ' ';   /* discard new-line character at end of line */
        result = pop();
```

```
        if (!is_empty()) {
          printf("*** Malformed expression (not enough operators) ***\n\n");
          make_empty();
          continue;
        }
        printf("Value: %d\n\n", result);
        break;

      case 1:
        if (!isdigit(token[0])) {   /* must be an operator */
          if (is_empty()) {
            error("Malformed expression (not enough operands)");
            continue;
          }
          op1 = pop();
          if (is_empty()) {
            error("Malformed expression (not enough operands)");
            continue;
          }
          op2 = pop();
          switch (token[0]) {
            case '+': push(op2 + op1);
                      break;
            case '-': push(op2 - op1);
                      break;
            case '*': push(op2 * op1);
                      break;
            case '/': push(op2 / op1);
                      break;
            default:  error("Illegal operator");
                      continue;
          }
          break;
        }
        /* FALL THROUGH */

      default:   /* operand is a number */
        if (is_full()) {
          error("Expression too complicated");
          continue;
        }
        if (!convert_to_number(token, &op1)) {
          error("Illegal operand");
          continue;
        }
        push(op1);
        break;
    }
  }
}

int read_token(char *token, int n)
{
  /* Assumptions: Tokens are separated by one or more blanks.
   *              Operands may begin with + or -.
   * Returns length of token (-1 indicates token too long, 0 indicates end
of line)
   */

  int i = 0;

  while (current_char == ' ')
    current_char = getchar();   /* skip blanks */
```

```
  while (current_char != ' ' && current_char != '\n') {
    if (i == n)
      return -1;   /* token too long */
    token[i++] = current_char;
    current_char = getchar();
  }

  token[i] = '\0';
  return i;
}

int convert_to_number(const char *token, int *value)
{
  /* No check for numeric overflow. Returns 1 if number is valid; 0
otherwise. */

  int i = 0, sign = 1;

  /* see if there's a sign */
  if (token[0] == '+')
    i++;
  else if (token[0] == '-') {
    sign = -1;
    i++;
  }

  *value = 0;
  do {
    if (!isdigit(token[i]))
      return 0;   /* not a number */
    *value = *value * 10 + token[i] - '0';
    i++;
  } while (token[i] != '\0');

  *value *= sign;   /* adjust for sign */

  return 1;
}

void error(const char *msg)
{
  char token[MAX_LEN+1];

  printf("*** %s ***\n\n", msg);
  while (read_token(token, MAX_LEN) != 0)
    ;   /* skip rest of line */
  current_char = ' ';   /* discard new-line character at end of line */
  make_empty();
}
```

Contents of `stack.c` file:

```
#include <stdio.h>
#include <stdlib.h>
#include "stack.h"

#define STACK_SIZE 100

int contents[STACK_SIZE];
int top = 0;
```

```
void make_empty(void)
{
  top = 0;
}

int is_empty(void)
{
  return top == 0;
}

int is_full(void)
{
  return top == STACK_SIZE;
}

void push(int i)
{
  if (is_full()) {
    printf("*** Stack overflow; program terminated. ***\n");
    exit(EXIT_FAILURE);
  }
  contents[top++] = i;
}

int pop(void)
{
  if (is_empty()) {
    printf("*** Stack underflow; program terminated. ***\n");
    exit(EXIT_FAILURE);
  }
  return contents[--top];
}
```

8. (a) `main.c`, `f1.c`, and `f2.c`.
(b) `f1.c` (assuming that `f1.h` is not affected by the change).
(c) `main.c`, `f1.c`, and `f2.c`, since all three include `f1.h`.
(d) `f1.c` and `f2.c`, since both include `f2.h`.

16 Structures, Unions, and Enumerations

2. (a)
```
struct {
  double re, im;
} c1, c2, c3;
```

(b)
```
struct {
  double re, im;
} c1 = {0.0, 1.0}, c2 = {1.0, 0.0}, c3;
```

(c) Only one statement is necessary:
```
c1 = c2;
```

(d)
```
c3.re = c1.re + c2.re;
c3.im = c1.im + c2.im;
```

4. (a)
```
typedef struct {
  double re, im;
} Complex;
```

(b)
```
Complex c1, c2, c3;
```

(c)
```
Complex make_complex(double real, double imag)
{
  Complex c;

  c.re = real;
  c.im = imag;
  return c;
}
```

(d)
```
Complex add_complex(Complex c1, Complex c2)
{
  Complex c3;

  c3.re = c1.re + c2.re;
  c3.im = c1.im + c2.im;
  return c3;
}
```

6.
```
#include <stdio.h>
#include <string.h>

#define COUNTRY_COUNT (sizeof(country_codes)/sizeof(country_codes[0]))

struct dialing_code {
  char *country;
  int code;
};

const struct dialing_code country_codes[] =
  {{"Argentina",           54}, {"Bangladesh",  880},
   {"Brazil",              55}, {"China",        86},
   {"Colombia",            57}, {"Egypt",        20},
   {"Ethiopia",           251}, {"France",       33},
```

```
    {"Germany",             49}, {"India",        91},
    {"Indonesia",           62}, {"Iran",         98},
    {"Italy",               39}, {"Japan",        81},
    {"Korea, Republic of",  82}, {"Mexico",       52},
    {"Nigeria",            234}, {"Pakistan",     92},
    {"Philippines",         63}, {"Poland",       48},
    {"Russia",               7}, {"South Africa", 27},
    {"Spain",               34}, {"Thailand",     66},
    {"Turkey",              90}, {"Ukraine",       7},
    {"United Kingdom",      44}, {"Vietnam",      84},
    {"Zaire",              243}};

main()
{
  char country_name[19];
  int i;

  printf("Enter country: ");
  scanf("%18s", country_name);

  for (i = 0; i < COUNTRY_COUNT; i++)
    if (strcmp(country_name, country_codes[i].country) == 0) {
      printf("\nThe dialing code for %s is %d\n", country_codes[i].country,
             country_codes[i].code);
      return 0;
    }

  printf("\nCountry not found\n");
  return 0;
}
```

8.

```
#include <stdio.h>
#include "readline.h"

#define NAME_LEN 25
#define MAX_PARTS 100

struct part {
  int number;
  char name[NAME_LEN+1];
  int on_hand;
};

int find_part(int number, const struct part inv[], int np);
void insert(struct part inv[], int *np);
void search(const struct part inv[], int np);
void update(struct part inv[], int np);
void print(const struct part inv[], int np);

/**********************************************************
 * main: Prompts the user to enter an operation code,     *
 *       then calls a function to perform the requested   *
 *       action. Repeats until the user enters the        *
 *       command 'q'. Prints an error message if the user *
 *       enters an illegal code.                          *
 **********************************************************/
main()
{
  char code;
  struct part inventory[MAX_PARTS];
  int num_parts = 0;
```

```
  for (;;) {
    printf("Enter operation code: ");
    scanf(" %c", &code);
    while (getchar() != '\n')   /* skips to end of line */
      ;
    switch (code) {
      case 'i': insert(inventory, &num_parts);
                break;
      case 's': search(inventory, num_parts);
                break;
      case 'u': update(inventory, num_parts);
                break;
      case 'p': print(inventory, num_parts);
                break;
      case 'q': return 0;
      default:  printf("Illegal code\n");
    }
    printf("\n");
  }
}

/**********************************************************
 * find_part: Looks up a part number in the inventory     *
 *            array. Returns the array index if the part  *
 *            number is found; otherwise, returns -1.     *
 **********************************************************/
int find_part(int number, const struct part inv[], int np)
{
  int i;

  for (i = 0; i < np; i++)
    if (inv[i].number == number)
      return i;
  return -1;
}

/**********************************************************
 * insert: Prompts the user for information about a new   *
 *         part and then inserts the part into the        *
 *         database. Prints an error message and returns  *
 *         prematurely if the part already exists or the  *
 *         database is full.                              *
 **********************************************************/
void insert(struct part inv[], int *np)
{
  int part_number;

  if (*np == MAX_PARTS) {
    printf("Database is full; can't add more parts.\n");
    return;
  }

  printf("Enter part number: ");
  scanf("%d", &part_number);
  if (find_part(part_number, inv, *np) >= 0) {
    printf("Part already exists.\n");
    return;
  }

  inv[*np].number = part_number;
  printf("Enter part name: ");
  read_line(inv[*np].name, NAME_LEN);
  printf("Enter quantity on hand: ");
```

```
  scanf("%d", &inv[*np].on_hand);
  (*np)++;
}

/**********************************************************
 * search: Prompts the user to enter a part number, then  *
 *         looks up the part in the database. If the part *
 *         exists, prints the name and quantity on hand;  *
 *         if not, prints an error message.               *
 **********************************************************/
void search(const struct part inv[], int np)
{
  int i, number;

  printf("Enter part number: ");
  scanf("%d", &number);
  i = find_part(number, inv, np);
  if (i >= 0) {
    printf("Part name: %s\n", inv[i].name);
    printf("Quantity on hand: %d\n", inv[i].on_hand);
  } else
    printf("Part not found.\n");
}

/**********************************************************
 * update: Prompts the user to enter a part number.       *
 *         Prints an error message if the part doesn't    *
 *         exist; otherwise, prompts the user to enter    *
 *         change in quantity on hand and updates the     *
 *         database.                                      *
 **********************************************************/
void update(struct part inv[], int np)
{
  int i, number, change;

  printf("Enter part number: ");
  scanf("%d", &number);
  i = find_part(number, inv, np);
  if (i >= 0) {
    printf("Enter change in quantity on hand: ");
    scanf("%d", &change);
    inv[i].on_hand += change;
  } else
    printf("Part not found.\n");
}

/**********************************************************
 * print: Prints a listing of all parts in the database,  *
 *        showing the part number, part name, and         *
 *        quantity on hand. Parts are printed in the      *
 *        order in which they were entered into the       *
 *        database.                                       *
 **********************************************************/
void print(const struct part inv[], int np)
{
  int i;

  printf("Part Number   Part Name                  "
         "Quantity on Hand\n");
  for (i = 0; i < np; i++)
    printf("%7d       %-25s%11d\n", inv[i].number,
           inv[i].name, inv[i].on_hand);
}
```

10. The `a` member will occupy 4 bytes, the union `e` will take 4 bytes (the largest members, `b` and `c`, are both 4 bytes long), and the array `f` will require 4 bytes, so the total space allocated for `s` will be 12 bytes.

12. (a)
```
float area(struct shape s)
{
  if (s.shape_kind == RECTANGLE)
    return s.u.rectangle.length * s.u.rectangle.width;
  else
    return 3.14159 * s.u.circle.radius * s.u.circle.radius;
}
```

(b)
```
struct point center(struct shape s)
{
  return s.center;
}
```

(c)
```
struct shape move(struct shape s, int x, int y)
{
  struct shape new_shape;

  new_shape.center.x = s.center.x + x;
  new_shape.center.y = s.center.y + y;

  return new_shape;
}
```

(d)
```
Bool is_inside(struct shape s, struct point p)
{
  if (s.shape_kind == RECTANGLE)
    /* assumption: length lies along the x axis, width lies along the y axis */
    return (p.x >= s.center.x - 0.5 * s.u.rectangle.length) &&
           (p.x <= s.center.x + 0.5 * s.u.rectangle.length) &&
           (p.y >= s.center.y - 0.5 * s.u.rectangle.width)  &&
           (p.y <= s.center.y + 0.5 * s.u.rectangle.width);
  else
    return sqrt((p.x - s.center.x) * (p.x - s.center.x) +
                (p.y - s.center.y) * (p.y - s.center.y)) <= s.u.circle.radius;
}
```

14. (a)
```
enum week_days {MON, TUE, WED, THU, FRI, SAT, SUN};
```
(b)
```
typedef enum {MON, TUE, WED, THU, FRI, SAT, SUN} Week_days;
```

16. All the statements are legal, since C allows integers and enumeration values to be mixed without restriction. Only (a), (d), and (e) are safe. (b) is not meaningful if `i` has a value other than 0 or 1. (c) will not yield a meaningful result if `b` has the value 1.

2.

```
char *strdup(const char *s)
{
  char *temp = malloc(strlen(s)+1);

  if (temp == NULL)
    return NULL;

  strcpy(temp, s);
  return temp;
}
```

4.

```
#include <stdio.h>
#include <stdlib.h>
#include "readline.h"

#define NAME_LEN 25
#define INITIAL_PARTS 10

struct part {
  int number;
  char name[NAME_LEN+1];
  int on_hand;
};

int num_parts = 0;              /* number of parts currently stored */
int max_parts = INITIAL_PARTS;  /* current maximum size of the inventory array */
struct part *inventory;

int find_part(int number);
void insert(void);
void search(void);
void update(void);
void print(void);

/**********************************************************
 * main: Prompts the user to enter an operation code,     *
 *       then calls a function to perform the requested   *
 *       action. Repeats until the user enters the        *
 *       command 'q'. Prints an error message if the user *
 *       enters an illegal code.                          *
 **********************************************************/
main()
{
  char code;

  inventory = malloc(max_parts * sizeof(struct part));
  if (inventory == NULL) {
    printf("Can't allocate initial inventory space.\n");
    exit(EXIT_FAILURE);
  }

  for (;;) {
    printf("Enter operation code: ");
    scanf(" %c", &code);
    while (getchar() != '\n')   /* skips to end of line */
      ;
```

```
    switch (code) {
      case 'i': insert();
                break;
      case 's': search();
                break;
      case 'u': update();
                break;
      case 'p': print();
                break;
      case 'q': return 0;
      default:  printf("Illegal code\n");
    }
    printf("\n");
  }
}

/*********************************************************
 * find_part: Looks up a part number in the inventory     *
 *            array. Returns the array index if the part  *
 *            number is found; otherwise, returns -1.     *
 *********************************************************/
int find_part(int number)
{
  int i;

  for (i = 0; i < num_parts; i++)
    if (inventory[i].number == number)
      return i;
  return -1;
}

/*********************************************************
 * insert: Prompts the user for information about a new   *
 *         part and then inserts the part into the        *
 *         database. Prints an error message and returns  *
 *         prematurely if the part already exists or the  *
 *         database is full.                              *
 *********************************************************/
void insert(void)
{
  int part_number;
  struct part *temp;

  if (num_parts == max_parts) {
    max_parts *= 2;
    temp = realloc(inventory, max_parts * sizeof(struct part));
    if (temp == NULL) {
      printf("Insufficient memory; can't add more parts.\n");
      return;
    }
    inventory = temp;
  }

  printf("Enter part number: ");
  scanf("%d", &part_number);
  if (find_part(part_number) >= 0) {
    printf("Part already exists.\n");
    return;
  }

  inventory[num_parts].number = part_number;
  printf("Enter part name: ");
  read_line(inventory[num_parts].name, NAME_LEN);
```

```
  printf("Enter quantity on hand: ");
  scanf("%d", &inventory[num_parts].on_hand);
  num_parts++;
}

/**********************************************************
 * search: Prompts the user to enter a part number, then  *
 *         looks up the part in the database. If the part *
 *         exists, prints the name and quantity on hand;  *
 *         if not, prints an error message.               *
 **********************************************************/
void search(void)
{
  int i, number;

  printf("Enter part number: ");
  scanf("%d", &number);
  i = find_part(number);
  if (i >= 0) {
    printf("Part name: %s\n", inventory[i].name);
    printf("Quantity on hand: %d\n", inventory[i].on_hand);
  } else
    printf("Part not found.\n");
}

/**********************************************************
 * update: Prompts the user to enter a part number.       *
 *         Prints an error message if the part doesn't    *
 *         exist; otherwise, prompts the user to enter    *
 *         change in quantity on hand and updates the     *
 *         database.                                      *
 **********************************************************/
void update(void)
{
  int i, number, change;

  printf("Enter part number: ");
  scanf("%d", &number);
  i = find_part(number);
  if (i >= 0) {
    printf("Enter change in quantity on hand: ");
    scanf("%d", &change);
    inventory[i].on_hand += change;
  } else
    printf("Part not found.\n");
}

/**********************************************************
 * print: Prints a listing of all parts in the database,  *
 *        showing the part number, part name, and         *
 *        quantity on hand. Parts are printed in the      *
 *        order in which they were entered into the       *
 *        database.                                       *
 **********************************************************/
void print(void)
{
  int i;

  printf("Part Number   Part Name                  "
         "Quantity on Hand\n");
  for (i = 0; i < num_parts; i++)
    printf("%7d       %-25s%11d\n", inventory[i].number,
           inventory[i].name, inventory[i].on_hand);
}
```

6. (b) and (c) are legal. (a) is illegal because it tries to reference a member of `d` without mentioning `d`. (d) is illegal because it uses `->` instead of `.` to reference the `c` member of `d`.

8. The first call of `free` will release the space for the first node in the list, making `p` a dangling pointer. Executing `p = p->next` to advance to the next node will have an undefined effect. Here's a correct way to write the loop, using a temporary pointer that points to the node being deleted:

```
struct node *temp;

p = first;
while (p != NULL) {
  temp = p;
  p = p->next;
  free(temp);
}
```

10.

```
#include <stdio.h>
#include <stdlib.h>
#include "stack.h"

#define TRUE 1
#define FALSE 0

struct node {
  int value;
  struct node *next;
};

struct node *top = NULL;

void make_empty(void)
{
  struct node *temp;

  while (top != NULL) {
    temp = top;
    top = top->next;
    free(temp);
  }
}

int is_empty(void)
{
  return top == NULL;
}

int push(int i)
{
  struct node *new_node;

  new_node = malloc(sizeof(struct node));
  if (new_node == NULL)
    return FALSE;

  new_node->value = i;
  new_node->next = top;
  top = new_node;

  return TRUE;
}
```

```
int pop(void)
{
  struct node *temp;
  int i;

  if (is_empty()) {
    printf("*** Stack underflow; program terminated. ***\n");
    exit(EXIT_FAILURE);
  }

  i = top->value;
  temp = top;
  top = top->next;
  free(temp);

  return i;
}
```

12. The output of the program is

```
Answer: 3
```

The program tests the values of `f2(0)`, `f2(1)`, `f2(2)`, and so on, stopping when `f2` returns zero. It then prints the argument that was passed to `f2` to make it return zero.

14. Assuming that `compare` is the name of the comparison function, the following call of `qsort` will sort the last 50 elements of `a`:

```
qsort(&a[50], 50, sizeof(a[0]), compare);
```

16.

```
#include <stdio.h>
#include <stdlib.h>
#include "readline.h"

#define NAME_LEN 25
#define MAX_PARTS 100

struct part {
  int number;
  char name[NAME_LEN+1];
  int on_hand;
} inventory[MAX_PARTS];

int num_parts = 0;   /* number of parts currently stored */

int find_part(int number);
void insert(void);
void search(void);
void update(void);
void print(void);
int compare_parts(const void *p, const void *q);

/*********************************************************
 * main: Prompts the user to enter an operation code,     *
 *       then calls a function to perform the requested   *
 *       action. Repeats until the user enters the        *
 *       command 'q'. Prints an error message if the user *
 *       enters an illegal code.                          *
 *********************************************************/
main()
{
  char code;
```

```
  for (;;) {
    printf("Enter operation code: ");
    scanf(" %c", &code);
    while (getchar() != '\n')   /* skips to end of line */
      ;
    switch (code) {
      case 'i': insert();
                break;
      case 's': search();
                break;
      case 'u': update();
                break;
      case 'p': print();
                break;
      case 'q': return 0;
      default:  printf("Illegal code\n");
    }
    printf("\n");
  }
}

/**********************************************************
 * find_part: Looks up a part number in the inventory     *
 *            array. Returns the array index if the part  *
 *            number is found; otherwise, returns -1.     *
 **********************************************************/
int find_part(int number)
{
  int i;

  for (i = 0; i < num_parts; i++)
    if (inventory[i].number == number)
      return i;
  return -1;
}

/**********************************************************
 * insert: Prompts the user for information about a new   *
 *         part and then inserts the part into the        *
 *         database. Prints an error message and returns  *
 *         prematurely if the part already exists or the  *
 *         database is full.                              *
 **********************************************************/
void insert(void)
{
  int part_number;

  if (num_parts == MAX_PARTS) {
    printf("Database is full; can't add more parts.\n");
    return;
  }

  printf("Enter part number: ");
  scanf("%d", &part_number);
  if (find_part(part_number) >= 0) {
    printf("Part already exists.\n");
    return;
  }

  inventory[num_parts].number = part_number;
  printf("Enter part name: ");
  read_line(inventory[num_parts].name, NAME_LEN);
  printf("Enter quantity on hand: ");
  scanf("%d", &inventory[num_parts].on_hand);
  num_parts++;
}
```

```
/*********************************************************
 * search: Prompts the user to enter a part number, then  *
 *         looks up the part in the database. If the part *
 *         exists, prints the name and quantity on hand;  *
 *         if not, prints an error message.               *
 *********************************************************/
void search(void)
{
  int i, number;

  printf("Enter part number: ");
  scanf("%d", &number);
  i = find_part(number);
  if (i >= 0) {
    printf("Part name: %s\n", inventory[i].name);
    printf("Quantity on hand: %d\n", inventory[i].on_hand);
  } else
    printf("Part not found.\n");
}

/*********************************************************
 * update: Prompts the user to enter a part number.       *
 *         Prints an error message if the part doesn't    *
 *         exist; otherwise, prompts the user to enter    *
 *         change in quantity on hand and updates the     *
 *         database.                                      *
 *********************************************************/
void update(void)
{
  int i, number, change;

  printf("Enter part number: ");
  scanf("%d", &number);
  i = find_part(number);
  if (i >= 0) {
    printf("Enter change in quantity on hand: ");
    scanf("%d", &change);
    inventory[i].on_hand += change;
  } else
    printf("Part not found.\n");
}

/*********************************************************
 * print: Prints a listing of all parts in the database,  *
 *        showing the part number, part name, and         *
 *        quantity on hand. Parts are printed in the      *
 *        order in which they were entered into the       *
 *        database.                                       *
 *********************************************************/
void print(void)
{
  int i;

  qsort(inventory, num_parts, sizeof(struct part), compare_parts);
  printf("Part Number   Part Name                  "
         "Quantity on Hand\n");
  for (i = 0; i < num_parts; i++)
    printf("%7d       %-25s%11d\n", inventory[i].number,
           inventory[i].name, inventory[i].on_hand);
}

int compare_parts(const void *p, const void *q)
{
  return ((struct part *) p)->number - ((struct part *) q)->number;
}
```

18 Declarations

2. (a) `extern`
(b) `static`
(c) `extern` and `static` (when applied to a local variable)

4. If `f` has never been called previously, the value of `f(10)` will be 0. If `f` has been called five times previously, the value of `f(10)` will be 50, since `j` is incremented once per call.

6. (a) `x` is an array of ten pointers to functions. Each function takes an `int` argument and returns a character.
(b) `x` is a function that returns a pointer to an array of five integers.
(c) `x` is a function with no arguments that returns a pointer to a function with an `int` argument that returns a pointer to an array of ten `float` values.
(d) `x` is a function with two arguments. The first argument is an integer, and the second is a pointer to a function with an `int` argument and no return value. `x` returns a pointer to a function with an `int` argument and no return value. (Although this example may seem artificially complex, the `signal` function—part of the standard C library—has exactly this prototype. See p. 543 for a discussion of `signal`.)

8. (a)
```
char *(*p)(char *);
```
(b)
```
void *f(struct t *p, long int n)(void);
```
(c)
```
void (*a[])(void) = {insert, search, update, print};
```
(d)
```
struct t (*b[10])(int, int);
```

10. (a), (c), and (d) are legal. (b) is illegal; the initializer for a variable with static storage duration must be a constant expression, and `i * i` doesn't qualify.

12. (a). Variables with static storage duration are initialized to zero by default; variables with automatic storage duration have no default initial value.

2.

```
#include <stdio.h>
#include <stdlib.h>
#include "stack.h"

struct node {
  int data;
  struct node *next;
};

PRIVATE struct node *top = NULL;

PUBLIC void make_empty(void)
{
  top = NULL;
}

PUBLIC int is_empty(void)
{
  return top == NULL;
}

PUBLIC void push(int i)
{
  struct node *new_node;

  new_node = malloc(sizeof(struct node));
  if (new_node == NULL) {
    printf("Error in push: stack is full.\n");
    exit(EXIT_FAILURE);
  }

  new_node->data = i;
  new_node->next = top;
  top = new_node;
}

PUBLIC int pop(void)
{
  struct node *old_top;
  int i;

  if (is_empty()) {
    printf("Error in pop: stack is empty.\n");
    exit(EXIT_FAILURE);
  }

  old_top = top;
  i = top->data;
  top = top->next;
  free(old_top);
  return i;
}
```

4. (a) Contents of `stack.c` file:

```
#include <stdio.h>
#include <stdlib.h>
#include "stack.h"

void make_empty(Stack *s)
{
  s->top = 0;
}

int is_empty(const Stack *s)
{
  return s->top == 0;
}

void push(Stack *s, int i)
{
  if (s->top == STACK_SIZE) {
    printf("Error in push: stack is full.\n");
    exit(EXIT_FAILURE);
  }
  s->contents[s->top++] = i;
}

int pop(Stack *s)
{
  if (is_empty(s)) {
    printf("Error in pop: stack is empty.\n");
    exit(EXIT_FAILURE);
  }
  return s->contents[--s->top];
}
```

(b) Contents of `stack.h` file:

```
#ifndef STACK_H
#define STACK_H

struct node {
  int data;
  struct node *next;
};

typedef struct node *Stack;

void make_empty(Stack *s);
int is_empty(Stack s);
void push(Stack *s, int i);
int pop(Stack *s);

#endif
```

Contents of `stack.c` file:

```
#include <stdio.h>
#include <stdlib.h>
#include "stack.h"

void make_empty(Stack *s)
{
  *s = NULL;
}
```

```
int is_empty(Stack s)
{
  return s == NULL;
}

void push(Stack *s, int i)
{
  struct node *new_node;

  new_node = malloc(sizeof(struct node));
  if (new_node == NULL) {
    printf("Error in push: stack is full.\n");
    exit(EXIT_FAILURE);
  }

  new_node->data = i;
  new_node->next = *s;
  *s = new_node;
}

int pop(Stack *s)
{
  struct node *old_top;
  int i;

  if (is_empty(*s)) {
    printf("Error in pop: stack is empty.\n");
    exit(EXIT_FAILURE);
  }

  old_top = *s;
  i = (*s)->data;
  *s = (*s)->next;
  free(old_top);
  return i;
}
```

6. No. `Fraction` objects do not contain pointers to dynamically allocated memory, so no special action has to be taken when a `Fraction` object ceases to exist.

8. Using << and >> instead of `printf` and `scanf` helps prevent many common C errors, including: (1) putting the wrong number of conversion specifications in a format string, (2) using an incorrect conversion specification, and (3) forgetting to include the address operator in a `scanf` call. Another advantage is that << and >> can be extended to read and write new classes, whereas the behavior of `printf` and `scanf` can't be changed.

10. The following template for the `Queue` class assumes that `Bool` is a Boolean type defined elsewhere. The member functions return a true value to indicate success, and a false value to indicate failure.

```
template <class T>
class Queue {
public:
  Bool insert(T);
  Bool remove(T&);
  Bool first(T&);
  Bool last(T&);
  Bool isEmpty();
private:
  // data members to implement Queue
};
```

The definition of the `insert` function will have the following form; the other member functions are defined in a similar fashion.

```
template <class T>
Bool Queue<T>::insert(T x)
{
  …
}
```

20 Low-Level Programming

2. To toggle a bit in a variable `i`, apply the exclusive-or operator (^) to `i` and a mask with a 1 bit in the desired position, then store the result back into `i`. To toggle bit 4, for example, use the statement

```
i = i ^ 0x0010;
```

or, more concisely,

```
i ^= 0x0010;
```

4.
```
#define MK_COLOR(red, green, blue) ((long) (blue) << 16 | (green) << 8 | (red))
```

6. (a)
```
#include <stdio.h>

unsigned short int swap_bytes(unsigned short int i);

main()
{
  unsigned short int i;

  printf("Enter a hexadecimal number: ");
  scanf("%hx", &i);
  printf("Number with bytes swapped:  %hx\n", swap_bytes(i));
  return 0;
}

unsigned short int swap_bytes(unsigned short int i)
{
  unsigned short int high_byte = i << 8;
  unsigned short int low_byte = i >> 8;

  return high_byte | low_byte;
}
```

(b)
```
unsigned short int swap_bytes(unsigned short int i)
{
  return i << 8 | i >> 8;
}
```

8. (a) The value of `~0` is a number containing all 1 bits. Shifting this number to the left by `n` places yields a result of the form 1…10…0, where there are `n` 0 bits. Applying the ~ operator to that number yields a result of the form 0…01…1, where there are `n` 1 bits.

(b) It returns a bit-field within `i` of length `n` starting at position `m`. Positions are assumed to be numbered starting from the rightmost bit, which is position 0.

10.
```
#include <stdio.h>

struct IEEE_float {
  unsigned int fraction: 23;   /* members may need to be reversed */
  unsigned int exponent: 8;
  unsigned int sign: 1;
};
```

```
union dual_float {
  float flt;
  struct IEEE_float IEEE;
};

main()
{
  union dual_float dual;

  dual.IEEE.sign = 1;
  dual.IEEE.exponent = 128;
  dual.IEEE.fraction = 0;

  printf("Value: %g\n", dual.flt);

  return 0;
}
```

2. In the header files that come with one compiler, the following functions are hidden by macros. (Your mileage may vary, of course.)

   `<ctype.h>`: `isalnum`, `isalpha`, `iscntrl`, `isdigit`, `isgraph`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `isxdigit`

   `<stdio.h>`: `ferror`, `feof`, `ungetc`

   `<stdlib.h>`: `atoi`

4. (a) `<time.h>`
   (b) `<ctype.h>`
   (c) `<limits.h>`
   (d) `<math.h>`
   (e) `<limits.h>`
   (f) `<float.h>`
   (g) `<string.h>`
   (h) `<stdio.h>`

22 Input/Output

2. (a) `"rb+"`
(b) `"a"`
(c) `"rb"`
(d) `"r"`

4. (a) `00000083.736`
(b) `00000029749.`
(c) `001.0549e+09`
(d) `002.3522e-05`

6.
```
printf(widget == 1 ? "%d widget" : "%d widgets", widget);
```

8. No. The difference is that `"%1s"` will store a null character after it reads and stores a nonblank character; `" %c"` will store only the nonblank character. As a result, the two format strings must be used differently:

```
char c, s[2];

scanf(" %c", &c);  /* stores a nonblank character into c */
scanf("%1s", s);   /* stores a nonblank character into s[0]
                      and a null character into s[1] */
```

10. Revise the program's `while` loop as follows:

```
while ((ch = getc(source_fp)) != EOF) {
  if (putc(ch, dest_fp) == EOF) {
    fprintf(stderr, "Error during writing; copy terminated prematurely\n");
    exit(EXIT_FAILURE);
  }
}
```

12.
```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>

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

  if (argc != 2) {
    fprintf(stderr, "usage: toupper file\n");
    exit(EXIT_FAILURE);
  }

  if ((fp = fopen(argv[1], "r")) == NULL) {
    fprintf(stderr, "Can't open %s\n", argv[1]);
    exit(EXIT_FAILURE);
  }

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

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

14. (a)
```
#include <stdio.h>
#include <stdlib.h>

main(int argc, char *argv[])
{
  FILE *fp;
  int ch, count = 0;

  if (argc != 2) {
    fprintf(stderr, "usage: cntchar file\n");
    exit(EXIT_FAILURE);
  }

  if ((fp = fopen(argv[1], "r")) == NULL) {
    fprintf(stderr, "Can't open %s\n", argv[1]);
    exit(EXIT_FAILURE);
  }

  while((ch = getc(fp)) != EOF)
    count++;

  printf("There are %d characters in %s\n", count, argv[1]);

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

(b)
```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>

main(int argc, char *argv[])
{
  FILE *fp;
  int ch, count = 0;
  enum {FALSE, TRUE} in_word;

  if (argc != 2) {
    fprintf(stderr, "usage: cntword file\n");
    exit(EXIT_FAILURE);
  }

  if ((fp = fopen(argv[1], "r")) == NULL) {
    fprintf(stderr, "Can't open %s\n", argv[1]);
    exit(EXIT_FAILURE);
  }

  in_word = FALSE;
  while ((ch = getc(fp)) != EOF)
    if (isspace(ch))
      in_word = FALSE;
    else if (!in_word) {
      in_word = TRUE;
      count++;
    }

  printf("There are %d words in %s\n", count, argv[1]);

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

(c)
```
#include <stdio.h>
#include <stdlib.h>

main(int argc, char *argv[])
{
  FILE *fp;
  int ch, count = 0;

  if (argc != 2) {
    fprintf(stderr, "usage: cntline file\n");
    exit(EXIT_FAILURE);
  }

  if ((fp = fopen(argv[1], "r")) == NULL) {
    fprintf(stderr, "Can't open %s\n", argv[1]);
    exit(EXIT_FAILURE);
  }

  while((ch = getc(fp)) != EOF)
    if (ch == '\n')
      count++;

  printf("There are %d lines in %s\n", count, argv[1]);

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

16.
```
#include <stdio.h>
#include <stdlib.h>

main(int argc, char *argv[])
{
  FILE *fp;
  int ch, count = 0;

  if (argc != 2) {
    fprintf(stderr, "usage: hexdump file\n");
    exit(EXIT_FAILURE);
  }

  if ((fp = fopen(argv[1], "rb")) == NULL) {
    fprintf(stderr, "Can't open %s\n", argv[1]);
    exit(EXIT_FAILURE);
  }

  while ((ch = getc(fp)) != EOF) {
    if (count == 20) {
      printf("\n");
      count = 0;
    }
    printf("%.2x ", ch);
    count++;
  }

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

18. (a)
```
char *fget_string(char *s, int n, FILE *stream)
{
  int ch, len = 0;

  while (len < n - 1) {
    if ((ch = getc(stream)) == EOF) {
      if (len == 0 || ferror(stream))
        return NULL;
      break;
    }
    s[len++] = ch;
    if (ch == '\n')
      break;
  }

  s[len] = '\0';
  return s;
}
```

(b)
```
int fput_string(const char *s, FILE *stream)
{
  while (*s != '\0') {
    if (putc(*s, stream) == EOF)
      return EOF;
    s++;
  }

  return 0;
}
```

20.
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define NAME_LEN 25

struct part {
  int number;
  char name[NAME_LEN + 1];
  int on_hand;
};

main(int argc, char *argv[])
{
  FILE *in_fp1, *in_fp2, *out_fp;
  int num_read1, num_read2;
  struct part part1, part2;

  if (argc != 4) {
    fprintf(stderr, "usage: merge infile1 infile2 outfile\n");
    exit(EXIT_FAILURE);
  }

  if ((in_fp1 = fopen(argv[1], "rb")) == NULL) {
    fprintf(stderr, "Can't open %s\n", argv[1]);
    exit(EXIT_FAILURE);
  }

  if ((in_fp2 = fopen(argv[2], "rb")) == NULL) {
    fprintf(stderr, "Can't open %s\n", argv[2]);
    exit (EXIT_FAILURE);
  }
```

```
  if ((out_fp = fopen(argv[3], "wb")) == NULL) {
    fprintf(stderr, "Can't open %s\n", argv[3]);
    exit(EXIT_FAILURE);
  }

  num_read1 = fread(&part1, sizeof(struct part), 1, in_fp1);
  num_read2 = fread(&part2, sizeof(struct part), 1, in_fp2);
  while (num_read1 == 1 && num_read2 == 1)
    /* successfully read records from both files */
    if (part1.number < part2.number) {
      fwrite(&part1, sizeof(struct part), 1, out_fp);
      num_read1 = fread(&part1, sizeof(struct part), 1, in_fp1);
    } else if (part1.number > part2.number) {
      fwrite(&part2, sizeof(struct part), 1, out_fp);
      num_read2 = fread(&part2, sizeof(struct part), 1, in_fp2);
    } else {
      /* part numbers are equal */
      if (strcmp(part1.name, part2.name) != 0)
        printf("Names don't match for part %d; using the name %s\n",
               part1.number, part1.name);
      part1.on_hand += part2.on_hand;
      fwrite(&part1, sizeof(struct part), 1, out_fp);
      num_read1 = fread(&part1, sizeof(struct part), 1, in_fp1);
      num_read2 = fread(&part2, sizeof(struct part), 1, in_fp2);
    }

  /* copy rest of file1 to output file */
  while (num_read1 == 1) {
    fwrite(&part1, sizeof(struct part), 1, out_fp);
    num_read1 = fread(&part1, sizeof(struct part), 1, in_fp1);
  }

  /* copy rest of file2 to output file */
  while (num_read2 == 1) {
    fwrite(&part2, sizeof(struct part), 1, out_fp);
    num_read2 = fread(&part2, sizeof(struct part), 1, in_fp2);
  }

  fclose(in_fp1);
  fclose(in_fp2);
  fclose(out_fp);
  return 0;
}
```

22. (a) `fseek(fp, n * 64L, SEEK_SET);`
(b) `fseek(fp, -64L, SEEK_END);`
(c) `fseek(fp, 64L, SEEK_CUR);`
(d) `fseek(fp, -128L, SEEK_CUR);`

2.
```
double round(double x, int n)
{
  double power = pow(10.0, n);

  if (x < 0.0)
    return ceil(x * power - 0.5) / power;
  else
    return floor(x * power + 0.5) / power;
}
```

4.
```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>

#define BUFFER_LEN 1000
   /* limits the number of white-space characters at the beginning of a line */

main()
{
  int ch, n = 0;
  char buffer[BUFFER_LEN+1];
  enum {FALSE, TRUE} nonblank_seen = FALSE;

  while ((ch = getchar()) != EOF) {
    if (nonblank_seen)
      putchar(ch);
    else if (isspace(ch)) {
      if (n == BUFFER_LEN) {
        fprintf(stderr, "Buffer overflow; program terminated\n");
        exit(EXIT_FAILURE);
      }
      buffer[n++] = ch;
    } else {
      nonblank_seen = TRUE;
      buffer[n] = '\0';
      fputs(buffer, stdout);
      putchar(ch);
    }
    if (ch == '\n') {
      n = 0;
      nonblank_seen = FALSE;
    }
  }

  return 0;
}
```

6. (a) `memmove`
(b) `memmove` (we can't use `strcpy` because its behavior is undefined when the source of the copy overlaps with the destination)
(c) `strncpy`
(d) `memcpy`

8.

```
int numchar(const char *s, char ch)
{
  int count = 0;

  s = strchr(s, ch);
  while (s != NULL) {
    count++;
    s = strchr(s+1, ch);
  }

  return count;
}
```

10. `if (strstr("foo#bar#baz", str) != NULL)...`

The assumptions are that `str` is at least three characters long and doesn't contain the # character.

12. `memset(&s[strlen(s)-n], '!', n);`

24 Error Handling

2. (a)
```
double try_math_fcn(double (*f)(double), double x, const char *msg)
{
  double result;

  errno = 0;
  result = (*f)(x);
  if (errno != 0) {
    perror(msg);
    exit(EXIT_FAILURE);
  }
  return result;
}
```

(b)
```
#define TRY_MATH_FCN(f, x) try_math_fcn(f, x, "Error in call of " #f)
```

4.
```
main()
{
  char code;

  for (;;) {
    setjmp(env);
    printf("Enter operation code: ");
    scanf(" %c", &code);
    while (getchar() != '\n')   /* skips to end of line */
      ;
    switch (code) {
      case 'i': insert();
                break;
      case 's': search();
                break;
      case 'u': update();
                break;
      case 'p': print();
                break;
      case 'q': return 0;
      default:  printf("Illegal code\n");
    }
    printf("\n");
  }
}
```

The `jmp_buf` variable `env` will need to be global, rather than local to `main`, so that the function performing the `longjmp` will be able to supply it as an argument.

2.

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

main()
{
  char *temp, *C_locale;

  temp = setlocale(LC_ALL, NULL);   /* "C" is the current locale by default */
  if (temp == NULL) {
    printf("\"C\" locale information not available\n");
    return 0;
  }

  C_locale = malloc(strlen(temp)+1);
  if (C_locale == NULL) {
    printf("Can't allocate space to store locale information\n");
    return 0;
  }

  strcpy(C_locale, temp);

  temp = setlocale(LC_ALL, "");
  if (temp == NULL) {
    printf("Native locale information not available\n");
    return 0;
  }

  if (strcmp(temp, C_locale) == 0)
    printf("Native locale is the same as the \"C\" locale\n");
  else
    printf("Native locale is not the same as the \"C\" locale\n");

  return 0;
}
```

4.

```
while ((orig_char = getchar()) != EOF) ??<
  new_char = orig_char ??' KEY;
  if (iscntrl(orig_char) ??!??! iscntrl(new_char))
    putchar(orig_char);
  else
    putchar(new_char);
??>
```

2.
```
void int_printf(const char *format, ...)
{
  va_list ap;
  const char *p;
  int digit, i, power, temp;

  va_start(ap, format);

  for (p = format; *p != '\0'; p++) {
    if (*p != '%') {
      putchar(*p);
      continue;
    }

    if (*++p == 'd') {
      i = va_arg(ap, int);
      if (i < 0) {
        i = -i;
        putchar('-');
      }

      temp = i;
      power = 1;
      while (temp > 9) {
        temp /= 10;
        power *= 10;
      }

      do {
        digit = i / power;
        putchar(digit + '0');
        i -= digit * power;
        power /= 10;
      } while (i > 0);
    }
  }

  va_end(ap);
}
```

4. The statement converts the string that `p` points to into a long integer, storing the result in `value`. `p` is left pointing to the first character not included in the conversion.

6.
```
double rand_double(void)
{
  return (double) rand() / (RAND_MAX + 1);
}
```

8. (a)
```
#include <stdio.h>
#include <stdlib.h>

main()
{
  int count = 1000;
```

```
  while (count-- > 0)
    printf("%d", rand() & 1);

  printf("\n");
  return 0;
}
```

(b) For generating numbers in the range 0 to `N`–1, the formula `rand() / (RAND_MAX / N + 1)` often gives better results than `rand() % N`. For example, if `N` is 2 and `RAND_MAX` is 32,767, the formula works out to `rand() / 16,384`, which yields 0 if the value of `rand` is less than 16,384 and 1 if the value is greater than or equal to 16,384.

10.

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

#define N 100

int compare_ints(const void *p, const void *q);

main()
{
  int a[N], i;
  clock_t start_clock;

  for (i = 0; i < N; i++)
    a[i] = N - i;

  start_clock = clock();
  qsort(a, N, sizeof(a[0]), compare_ints);

  printf("Time used to sort %d integers: %g sec.\n", N,
         (clock() - start_clock) / (double) CLOCKS_PER_SEC);

  return 0;
}

int compare_ints(const void *p, const void *q)
{
  return *(int *)p - *(int *)q;
}
```

12.

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

main()
{
  struct tm t;
  int n, year;

  /* initialize unused members */
  t.tm_sec = t.tm_min = t.tm_hour = 0;
  t.tm_isdst = -1;

  printf("Enter month (1-12): ");
  scanf("%d", &t.tm_mon);
  t.tm_mon--;

  printf("Enter day (1-31): ");
  scanf("%d", &t.tm_mday);
```

```
  printf("Enter year: ");
  scanf("%d", &year);
  t.tm_year = year - 1900;

  printf("Enter number of days in future: ");
  scanf("%d", &n);

  t.tm_mday += n;
  mktime(&t);
  printf("\nFuture date: %d/%d/%d\n", t.tm_mon + 1, t.tm_mday, t.tm_year + 1900);

  return 0;
}
```

14. (a)

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

main()
{
  time_t current = time(NULL);
  struct tm *ptr;
  char date_time[37];

  ptr = localtime(&current);
  strftime(date_time, sizeof(date_time), "%A, %B %d, %Y  %I:%M", ptr);
  printf("%s%c\n", date_time, ptr->tm_hour <= 11 ? 'a' : 'p');

  return 0;
}
```

(b)

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

main()
{
  time_t current = time(NULL);
  char date_time[22];

  strftime(date_time, sizeof(date_time), "%a, %d %b %y  %H:%M",
           localtime(&current));
  puts(date_time);

  return 0;
}
```

(c)

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

main()
{
  time_t current = time(NULL);
  struct tm *ptr;
  char date[9], time[12];

  ptr = localtime(&current);
  strftime(date, sizeof(date), "%m/%d/%y", ptr);
  strftime(time, sizeof(time), "%I:%M:%S %p", ptr);

  /* print date and time, suppressing leading zero in hours */
  printf("%s  %s\n", date, time[0] == '0' ? &time[1] : time);

  return 0;
}
```