Answers to Odd-Numbered Exercises

Return to top

Chapter 3
Chapter 4
Chapter 5
Chapter 6
Chapter 7
Chapter 8
Chapter 9
Chapter 10
Chapter 11
Chapter 12
Chapter 13
Chapter 14
Chapter 16

Chapter 3

3-3

```
Testing.......1...2..3

The output would all appear on the same line since a newline character is not printed until
the last printf call.
```

3-5

```
main (Void)                // should be lowercase v
(                          // should be a {
    INT  sum;              // should be lowercase int
    /*  COMPUTE RESULT     // Needs closing */
    sum = 25 + 37 = 19     // Needs ; at the end
    /* DISPLAY RESULTS //  // Needs */ to close comment
    printf ("The answer is %i\n" sum);   // Missing comma
    return 0;
}
```

Chapter 4

4-3

```
0996    Digits 8 and 9 invalid in octal constant
0x10.5  Decimal point not valid in integer constant
98.7U   Unsigned qualifier only valid for integers
1.2Fe-7 Can't use F and e together
0X0G1   G is not a valid hexadecimal digit
17777s  s is not a valid qualifier
15,000  Commas not allowed in constants
```

4-5

```
d = d
```

4-7

```
#include <stdio.h>

int main (void)
{
   double  result;

   result = (3.31e-8 * 2.01e-7) / (7.16e-6 + 2.01e-8);
   printf ("result = %g\n", result);
```

```
   return 0;
}
```

Chapter 5

5-3

```
#include <stdio.h>

int main (void)
{
    int     n, triangularNumber;

    printf ("TABLE OF TRIANGULAR NUMBERS\n\n");
    printf (" n    Sum from 1 to n\n");
    printf ("---  ---------------\n");

    for ( n = 5;  n <= 50;  n +=  5) {
        triangularNumber = n * (n + 1) / 2;
        printf ("%2i          %i\n", n, triangularNumber);
    }

    return 0;
}
```

5-5

```
#include <stdio.h>

int main (void)
{
    int   n, two_to_the_n;

    printf ("TABLE OF POWERS OF TWO\n\n");
    printf (" n     2 to the n\n");
    printf ("---    ----------\n");

    two_to_the_n = 1;

    for ( n = 0;  n <= 10;  ++n ) {
        printf ("%2i        %i\n", n, two_to_the_n);
        two_to_the_n *= 2;
    }

    return 0;
}
```

5-7

The decimal point in the field width causes leading zeroes to be displayed before numbers. This is discussed in more detail in Chapter 16. In this case, it causes a leading zero to appear if the number of cents entered is less than 10 (for example, $29.05).

5-9

```
// Program 5.2

/* Program to calculate the 200th triangular number
   Introduction of the for statement               */

#include <stdio.h>

int main (void)
{
     int  n, triangularNumber;

     triangularNumber = 0;
     n = 1;

     while ( n <= 200 ) {
```

```
            triangularNumber = triangularNumber + n;
            n = n + 1;
      }

      printf ("The 200th triangular number is %i\n", triangularNumber);

      return 0;
}

// ------------------------------------------------
// Program 5.3

// Program to generate a table of triangular numbers

#include <stdio.h>

int main (void)
{
      int  n, triangularNumber;

      printf ("TABLE OF TRIANGULAR NUMBERS\n\n");
      printf (" n    Sum from 1 to n\n");
      printf ("---   ---------------\n");

      triangularNumber = 0;
      n = 1;

      while ( n <= 10 ) {
            triangularNumber +=  n;
            printf (" %i           %i\n", n, triangularNumber);
            ++n;
      }

      return 0;
}


// ------------------------------------------------
// Program 5.4

#include <stdio.h>

int main (void)
{
      int  n, number, triangularNumber;

      printf ("What triangular number do you want? ");
      scanf  ("%i", &number);

      triangularNumber = 0;
      n = 1;

      while ( n <= number ) {
            triangularNumber += n;
            ++n;
      }

      printf ("Triangular number %i is %i\n", number, triangularNumber);

      return 0;
}


// ------------------------------------------------
// Program 5.5

#include <stdio.h>

int main (void)
{
      int  n, number, triangularNumber, counter;
```

```
      counter = 1;

      while ( counter <= 5 ) {
           printf ("What triangular number do you want? ");
           scanf  ("%i", &number);

           triangularNumber = 0;
           n = 1;

           while ( n <= number ) {
                triangularNumber += n;
                ++n;
           }

           printf ("Triangular number %i is %i\n\n", number, triangularNumber);

           ++counter;
      }

      return 0;
}
```

5-11

```
// Program to sum the digits in a number

#include <stdio.h>

int main (void)
{
    int  number, right_digit, sum = 0;

    printf ("Enter your number: ");
    scanf ("%i", &number);

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

    printf ("The sum of the digits is %i\n", sum);

    return 0;
}
```

Chapter 6

6-3

```
#include <stdio.h>

int main (void)
{
    int  n1, n2;

    printf ("Please enter two integers: ");
    scanf ("%i%i", &n1, &n2);

    if (n2 == 0)
       printf ("Division by zero.\n");
    else
       printf ("Result of %i / %i is %.3f\n", n1, n2, (float) n1 / n2);

    return 0;
}
```

6-5

A negative number entered for Program 5.9 causes each digit to print out as a negative number. For example, if -123 is entered, the output would look like this:

```
-3-2-1
```

Here is a corrected version of that program.

```
// Program to reverse the digits of a number (revised)

#include <stdio.h>
#include <stdbool.h>

int main (void)
{
    int   number, right_digit;
    bool  isNegative = false;

    printf ("Enter your number.\n");
    scanf ("%i", &number);

    /* if keyed-in number is negative, make it
       positive, but remember it was negative */

    if ( number < 0 ) {
        number = -number;
        isNegative = true;
    }

    do {
        right_digit = number % 10;
        printf ("%i", right_digit);
        number = number / 10;
    }
    while ( number != 0 );

    if (isNegative == true)
        printf ("-");

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

6-7

```
// Program to generate a table of prime numbers (revised)

#include <stdio.h>
#include <stdbool.h>

int main (void)
{

    int   p, d;
    bool  isPrime;

    // we start off knowing 2 is prime

    printf ("2  ");

    // now start testing odd numbers from 3

    for ( p = 3;  p <= 50;  p +=2 )
    {
        isPrime = true;

        // only test odd divisors

        for ( d = 3;  d < p && isPrime == true;  d += 2 )
          if ( p % d  ==  0 )
            isPrime = false;

        if ( isPrime != false )
          printf ("%i  ", p);
```

```
    }

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

Chapter 7

7-3

```
// Modified Program 7.2 -- uses break statement

#include <stdio.h>

int main (void)
{
    int  ratingCounters[11], i, response;

    for ( i = 1;  i <= 10;  ++i )
       ratingCounters[i] = 0;

    printf ("Enter your responses\n");
    printf ("When you are done, enter 999\n");

    while (1) {
      scanf ("%i", &response);

      if (response == 999)
          break;

      if ( response < 1 || response > 10 )
          printf ("Bad response: %i\n", response);
      else
          ++ratingCounters[response];
    }

    printf ("\n\nRating   Number of Responses\n");
    printf ("------   -------------------\n");

    for ( i = 1;  i <= 10;  ++i )
        printf ("%4i%14i\n", i, ratingCounters[i]);

    return 0;
}
```

Some people prefer not to use the break statement since it disrupts the normal sequential flow of a program's execution. You can rewrite this program to not use the break statement by replacing the while in the above program with the following equivalent code:

```
     do
     {
          scanf ("%i", &response);

          if (response != 999)
            if ( response < 1 || response > 10 )
                printf ("Bad response: %i\n", response);
            else
                ++rating_counters[response];
     }
     while ( response != 999 );
```

7-5

Each array element is calculated as the sum of the preceding array elements. This produces the following output:

```
                1 1 2 4 8 16 32 64 128 256
```

7-7

```
// Prime numbers generated with Sieve of Erasthothenes
```

```
int main (void)
{
        int  P[151], i, j;
        int  n = 150;

        for (i = 2; i <= n; ++i)
                P[i] = 0;

        i = 2;

        while (i <= n) {
                if (P[i] == 0)
                        printf ("%i  ", i);

                j = 1;

                while (i * j <= n) {
                        P[i * j] = 1;
                        ++j;
                }

                ++i;
        }

        return 0;
}
```

Chapter 8

8-3

Here is a modified square_root function, a sample main routine that calculates the square root of 3 with different values of epsilon, and the output from these calls.

```
// Function to compute the square root of a number

#include <stdio.h>

float  squareRoot (float x, float epsilon)
{
    float  guess   = 1.0;

    while  ( absoluteValue (guess * guess - x) >= epsilon )
       guess = ( x / guess + guess ) / 2.0;

    return guess;
}

int main (void)
{
    printf ("%f\n", squareRoot (3.0, .1));
    printf ("%f\n", squareRoot (3.0, .01));
    printf ("%f\n", squareRoot (3.0, .001));
    printf ("%f\n", squareRoot (3.0, .0001));

    return 0;
}

1.750000
1.732143
1.732143
1.732051
```

8-5

```
float  square_root (float x)
{
    float  guess   = 1.0;
    float  epsilon = .00001;

    while  ( absoluteValue((guess * guess) / x  - 1.0) >= epsilon )
```

```
        guess = ( x / guess + guess ) / 2.0;

    return guess;
}
```

8-7

```
long int  x_to_the_n (int x, int n)
{
    long int  result = 1;

    while (n > 0)
    {
        result *= x;
        --n;
    }

    return result;
}
```

8-9

```
int  lcm (int u, int v)
{
    int  gcd (int u, int v);

    if ( u < 0 || v < 0 )
       return 0;
    else
       return  u * v / gcd (u, v);
}
```

8-11

```
long int  array_sum (int  values [], int  n)
{
    int       i;
    long int  sum = 0;

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

    return sum;
}
```

8-13

```
// Sort an array of integers
// descending (order == -1) or ascending (order == 1) sort

#include <stdio.h>

void sort (int  a[], int  n, int  order)
{
    int  i, j, temp;

    for ( i = 0;  i < n - 1;  ++i )
        for ( j = i + 1;  j < n;  ++j )
            if ( (order == -1 && a[i] < a[j]) ||
                 (order == 1  && a[i] > a[j]) )
            {
                 temp = a[i];
                 a[i] = a[j];
                 a[j] = temp;
            }
}

int main (void)
{
    int  i;
    int  array[16] = { 34, -5, 6, 0, 12, 100, 56, 22,
                       44, -3, -9, 12, 17, 22, 6, 11 };
    void sort (int  a[], int  n, int  order);
```

```
    printf ("The array before the sort:\n");

    for ( i = 0;  i < 16;  ++i )
        printf ("%i ", array[i]);

    printf ("\n\nThe array in ascending order:\n");

    sort (array, 16, 1);

    for ( i = 0;  i < 16;  ++i )
        printf ("%i ", array[i]);

    printf ("\n\nThe array in descending order:\n");

    sort (array, 16, -1);

    for ( i = 0;  i < 16;  ++i )
        printf ("%i ", array[i]);

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

8-15

Here is a modified getNumberAndBase routine. The rest of the program remains unchanged.

```
void  getNumberAndBase (void)
{
    printf ("Number to be converted? ");
    scanf ("%li", &numberToConvert);

    do {
        printf ("Enter base value between 2 and 16: ");
        scanf ("%i", &base);
    }
    while ( base < 2  ||  base > 16 );
}
```

Chapter 9

9-3

```
#include <stdio.h>

struct time
{
    int  hour;
    int  minutes;
    int  seconds;
};

// calculate elapsed time (assume t2 is later than t1)

struct time  elapsed_time (struct time  t1, struct time  t2)
{

    struct time  result = { 0, 0, 0 };

    // first subtract the seconds

    result.seconds = t2.seconds - t1.seconds;

    // if seconds < 0, need to borrow one minute

    if (result.seconds < 0) {
        result.seconds += 60;
        --t2.minutes;
    }
```

```
    // now subtract the minutes

    result.minutes = t2.minutes - t1.minutes;

    // if minutes < 0, need to borrow one hour

    if (result.minutes < 0) {
        result.minutes += 60;
        --t2.hour;

    }

    // now subtract the hours

    result.hour = t2.hour - t1.hour;

    // if hour < 0, need to borrow one day (crossed midnight)

    if (result.hour < 0)
        result.hour += 24;

    return  result;
}

int main (void)
{
    struct time  elapsed_time (struct time  t1, struct time  t2);
    struct time  t1 = { 3, 45, 15 }, t2 = { 9, 44, 03 },
                 t3 = {22, 50, 59 }, t4 = { 7, 30,  0 };
    struct time  result;

    result = elapsed_time (t1, t2);
    printf ("Time between %.2i:%.2i:%.2i and %.2i:%.2i:%.2i "
        "is %.2i:%.2i:%.2i\n",
        t1.hour, t1.minutes, t1.seconds, t2.hour, t2.minutes,
        t2.seconds, result.hour, result.minutes, result.seconds);

    result = elapsed_time (t2, t1);
    printf ("Time between %.2i:%.2i:%.2i and %.2i:%.2i:%.2i "
        "is %.2i:%.2i:%.2i\n",
        t2.hour, t2.minutes, t2.seconds, t1.hour, t1.minutes,
        t1.seconds, result.hour, result.minutes, result.seconds);

    result = elapsed_time (t3, t4);
    printf ("Time between %.2i:%.2i:%.2i and %.2i:%.2i:%.2i "
        "is %.2i:%.2i:%.2i\n",
        t3.hour, t3.minutes, t3.seconds, t4.hour, t4.minutes,
        t4.seconds, result.hour, result.minutes, result.seconds);

    return 0;
}
```

Here is some sample output from the program:

```
Time between 03:45:15 and 09:44:03 is 05:58:48
Time between 09:44:03 and 03:45:15 is 18:01:12
Time between 22:50:59 and 07:30:00 is 08:39:01
```

9-5

```
struct dateAndTime  clockKeeper (struct dateAndTime  dt)
{
    struct time   timeUpdate (struct time  now);
    struct date   dateUpdate (struct date  today);

    dt.stime = timeUpdate (dt.stime);

    // see if we have to go to the next day

    if ( dt.stime.hour == 0  &&  dt.stime.minutes == 0  &&
```

```
          dt.stime.seconds == 0 )
        dt.sdate  = dateUpdate (dt.sdate);

    return  dt;
}

// Here is a sample main routine and output:

int main (void)
{
   struct dateAndTime   dt1 = { { 12, 31, 2004 }, { 23, 59, 59 } };
   struct dateAndTime   dt2 = { { 2,  28, 2008 }, { 23, 59, 58 } };

   printf ("Current date and time is %.2i/%.2i/%.2i "
           "%.2i:%.2i:%.2i\n",
            dt1.sdate.month, dt1.sdate.day, dt1.sdate.year,
            dt1.stime.hour,
            dt1.stime.minutes, dt1.stime.seconds);

   dt1 = clockKeeper (dt1);

   printf ("Updated date and time is %.2i/%.2i/%.2i "
           "%.2i:%.2i:%.2i\n\n",
           dt1.sdate.month, dt1.sdate.day, dt1.sdate.year,
           dt1.stime.hour, dt1.stime.minutes, dt1.stime.seconds);

   printf ("Current date and time is %.2i/%.2i/%.2i "
           "%.2i:%.2i:%.2i\n",
           dt2.sdate.month, dt2.sdate.day, dt2.sdate.year,
           dt2.stime.hour, dt2.stime.minutes, dt2.stime.seconds);

   dt2 = clockKeeper (dt2);

   printf ("Updated date and time is %.2i/%.2i/%.2i "
           "%.2i:%.2i:%.2i\n\n",
           dt2.sdate.month, dt2.sdate.day, dt2.sdate.year,
           dt2.stime.hour, dt2.stime.minutes, dt2.stime.seconds);

   printf ("Current date and time is %.2i/%.2i/%.2i "
           "%.2i:%.2i:%.2i\n",
           dt2.sdate.month, dt2.sdate.day, dt2.sdate.year,
           dt2.stime.hour, dt2.stime.minutes, dt2.stime.seconds);

   dt2 = clockKeeper (dt2);

   printf ("Updated date and time is %.2i/%.2i/%.2i "
        "%.2i:%.2i:%.2i\n\n",
        dt2.sdate.month, dt2.sdate.day, dt2.sdate.year,
        dt2.stime.hour, dt2.stime.minutes, dt2.stime.seconds);

   return 0;
}


Current date and time is 12/31/2004 23:59:59
Updated date and time is 01/01/2005 00:00:00

Current date and time is 02/28/2008 23:59:58
Updated date and time is 02/28/2008 23:59:59

Current date and time is 02/28/2008 23:59:59
Updated date and time is 02/29/2008 00:00:00
```

Chapter 10

10-3

There are many different ways to solve this problem. Here, I replaced the alphabetic function with a function called wordchar that considers not only alphabetic characters but also apostrophes as part of a word. We also added a function called numchar that considers digits, decimal points, commas, and dashes as part of a number. These definitions aren't

perfect, but they do work reasonably well without making the program overly complex.

```
// Function to determine if a character is part of a word

bool wordchar (const char  c)
{
    if  ( (c >= 'a'  &&  c <= 'z') || (c >= 'A'  &&  c <= 'Z') || c == '\'' )
        return true;
    else
        return false;
}

// Function to determine if a character is part of a number

bool  numchar (const char  c)
{

    if ( (c >= '0' && c <= '9') || c == '.' || c == ',' || c == '-' )
       return  true;
    else
       return  false;
}

// Function to count the number of words in a string

int  countWords (const char  string[])
{
    int  i,  wordCount = 0;
    bool lookingForWord = true, wordchar (const char c), numchar (const char c);

    for ( i = 0;  string[i] != '\0';  ++i )
        if ( wordchar (string[i])  || numchar (string[i]) ) {
            if ( lookingForWord ) {
                 ++wordCount;
                 lookingForWord = false;
            }
        }
        else
            lookingForWord = true;

    return wordCount;
}

// Here's a sample main routine and associated output:

int main (void)
{
    char  text1[] = "The sum of $552,227 and $-1,204.50 is $551,002.50";
    char  text2[] = "It isn't that I don't understand you.";
    int   countWords (const char  string[]);

    printf ("%s -- words = %i\n", text1, countWords (text1));
    printf ("%s -- words = %i\n", text2, countWords (text2));
}

The sum of $552,227 and $-1,204.50 is $551,002.50 -- words = 8
It isn't that I don't understand you. -- words = 7
```

10-5

```
// find s1 inside source, return index number if found, -1 if not found

int  findString (const char  source[], const char  s[])
{
    int  i, j, foundit = false;

    // try each character in source

    for ( i = 0;  source[i] != '\0'  &&  !foundit;  ++i ) {
        foundit = true;
```

```
        // now see if corresponding chars from s match

        for ( j = 0;  s[j] != '\0' &&  foundit;  ++j )
            if ( source[j + i] != s[j] || source[j + i] == '\0' )
              foundit = false;

        if (foundit)
           return i;
    }

    return -1;
}
```

10-7

```
/* insert string s into string source starting at i
   This function uses the stringLength function defined
   in the chapter.

   Note: this function assumes source is big enough
         to store the inserted string (dangerous!)   */

void  insertString (char  source[], char  s[], int  i)
{
   int  j, lenS, lenSource;

   /* first, find out how big the two strings are */

   lenSource = stringLength (source);
   lenS = stringLength (s);

   /* sanity check here -- note that i == lenSource
      effectively concatenates s onto the end of source */

   if (i > lenSource)
      return;

   /* now we have to move the characters in source
      down from the insertion point to make room for s.
      Note that we copy the string starting from the end
      to avoid overwriting characters in source.
      We also copy the terminating null (j starts at lenS)
      as well since the final result must be null-terminated */

   for ( j = lenSource;  j >= i;  --j )
      source [lenS + j] = source [j];

   /* we've made room, now copy s into source at the
      insertion point */

   for ( j = 0;  j < lenS; ++j )
       source [j + i] = s[j];
}
```

10-9

```
bool  replaceString (char  source [], char  s1[], char  s2[])
{
    int  index;

    // first locate s1 inside the source

    index = findString (source, s1);

    if ( index == -1 )
       return  false;

    // now delete s1 from the source

    removeString (source, index, stringLength (s1));

    // now insert the new string
```

```
    insertString (source, s2, index);

    return true;
}
```

10-11

```
#include <stdbool.h>

int  strToInt (const char  string[])
{
    int  i = 0, intValue, result = 0;
    int  negative = false;

    // test for leading minus sign

    if ( string[0] == '-') {
        negative = true;
        i = 1;
    }

    while ( string[i] >= '0' && string[i] <= '9' ) {
        intValue = string[i] - '0';
        result = result * 10 + intValue;
        ++i;
    }

    if ( negative )
        result  = -result;

    return result;
}
```

10-13

```
void  uppercase ( char str[] )
{
    int  i;

    for ( i = 0;  str[i] != '\0';  ++i )
       if ( str[i] >= 'a'  &&  str[i]  <= 'z' )
           str[i] = str[i] - 'a' + 'A';
}
```

Chapter 11

11-3

You can solve this problem by setting up a "dummy" structure variable called listHead, for example:

```
                struct entry  listHead;
```

and you can then set it pointing to the head of the list by assigning the next member of listHead to point to the actual first entry of the list:

```
                listHead.next = &entry1;
```

Now to insert a new entry called newEntry at the front of the list, you can write:

```
                insertEntry (&new_entry, &list_head);
```

11-5

```
struct dentry
{
    int            value;
    struct dentry  *next;
    struct dentry  *prev;
};
```

```
int main (void)
{
    struct dentry  n1, n2, n3, *lptr;
    int            i;

    n1.value = 100;
    n2.value = 200;
    n3.value = 300;

    n1.next = &n2;
    n2.next = &n3;
    n3.next = (struct dentry *) 0;

    n1.prev = (struct dentry *) 0;
    n2.prev = &n1;
    n3.prev = &n2;

    // forward search through list

    lptr = &n1;

    while ( lptr != 0 ) {

       printf ("%i ", lptr->value);
       lptr = lptr->next;
    }

    printf ("\n");

    // backward search through list

    lptr = &n3;

    while ( lptr != 0 ) {
       printf ("%i ", lptr->value);
       lptr = lptr->prev;
    }

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

Here's the output from the test program:

```
100 200 300
300 200 100
```

11-7

```
/* Sort an array of integers into ascending order
   (pointer version )  */

void sort (int  *a, int  n)
{
   int  *aptr1, *aptr2, temp;

   for ( aptr1 = a;  aptr1 < a + n - 1;  ++aptr1 )
       for ( aptr2 = aptr1 + 1;  aptr2 < a + n;  ++aptr2 )
           if ( *aptr1 > *aptr2 ) {
              temp = *aptr1;
              *aptr1 = *aptr2;
              *aptr2 = temp;
           }
}
```

11-9

```
/* Function to read a line of text from the terminal
   (pointer version)    */

void  readLine (char  *buffer)
```

```
{
    char  character;

    do {
        character = getchar ();
        *buffer++ = character;
    }
    while ( character != '\n' );

    *(buffer - 1) = '\0';
}
```

11-11

```
/* Function to calculate tomorrow's date
   (pointer version) */

void  dateUpdate (struct date  *today)
{
    int  numberOfDays (struct date  d);

    if ( today->day != numberOfDays (*today) )
        ++today->day;
    else if ( today->month == 12 ) {  /* end of year */
        today->day = 1;
        today->month = 1;
        ++today->year;
    }
    else {                            /* end of month */
        today->day = 1;
        ++today->month;
    }
}
```

Chapter 12

12-3

```
int  int_size (void)
{
    unsigned int  bits;
    int           size = 0;

    bits = ~0;

    while ( bits ) {
       ++size;
       bits >>= 1;
    }

    return size;
}
```

12-5

```
/* test bit n in word to see if it is on
   assumes words are 32 bits long        */

int  bit_test (unsigned int  word, int  n)
{
    if ( n < 0  || n > 31 )
       return  0;

    if ( (word >> (31 - n)) & 0x1 )
       return 1;
    else
       return 0;
}

unsigned int  bit_set (unsigned int  word, int  n)
```

```
{
    if ( n < 0  || n > 31 )
       return  0;

    return  word | (1 << (31 - n));
}
```

12-7

```
// extract count bits from value beginning at bit n

unsigned int bitpat_get (unsigned int value, int n, int count)
{
    int   word_size, i;

    word_size = int_size ();     // From exercise 3

    if ( n < 0  ||  n > word_size  || count < 0  ||  count + n > word_size )
       return  0;

    // first shift value to the leftmost part of the word

    value <<= n;

    // now when we shift right, the bits to the left of value will be cleared

    return  value  >>  word_size - count;
}
```

Chapter 13

13-3

```
#define MIN(x, y)   (((x) < (y)) ? (x) : (y))
```

13-5

```
#define SHIFT(value, n)  (((n) > 0) ? ((value) << (n))  \
                                    : ((value) >> -(n)))
```

13-7

```
#define IS_ALPHABETIC(c)  (IS_LOWER_CASE (c) || IS_UPPERCASE (c))
```

13-9

```
#define ABSOLUTE_VALUE(x)  (((x) < 0) ? -(x) : (x))
```

Chapter 14

14-1

```
typedef int (*FnPtr) (void);
```

14-3

```
Expression              Type    Value

f + i                   float   101
l / d                   double  33.3333
i / l + f               float   1.0
l * i                   long    50000
f / 2                   float   0.5
i / (d + f)             double  6.25
l / (i * 2.0)           double  2.5
l + i / (double) l      double  501.0
```

Chapter 16

16-3

```
// Program to copy one file to another
```

```
#include <stdio.h>

/* convert lowercase to uppercase character */

#define  TO_UPPER(c)  ((c >= 'a'  &&  c <= 'z') ? c - 'a' + 'A' : c)

int main (void)
{
    char  inName[64], outName[64];
    FILE  *in, *out;
    int   c;

    printf ("Enter name of file to be copied: ");
    scanf ("%63s", inName);
    printf ("Enter name of output file: ");
    scanf ("%63s", outName);

    if ( (in = (FILE *) fopen (inName, "r"))  ==  NULL ) {
       fprintf (stderr, "Can't open %s for reading.\n", inName);
       return 1;
    }
    else if ( (out = fopen (outName, "w"))  ==  NULL ) {
       fprintf (stderr, "Can't open %s for writing.\n", outName);
       return 2;
    }

    while ( (c = getc (in)) != EOF )
       putc (TO_UPPER (c), out);

    printf ("File has been copied.\n");

    return 0;
}
```

16-5

```
/* Program to extract columns from each line of a file
   (similar to the UNIX cut command) */

#include <stdio.h>

int main (void)
{

    char  inName[64];
    FILE  *in;
    int   m, n, curcol, c;

    printf ("Enter name of file: ");
    scanf ("%63s", inName);
    printf ("Enter starting and ending column numbers: ");
    scanf ("%i%i", &m, &n);

    if ( (in = fopen (inName, "r"))  ==  NULL ) {
        fprintf (stderr, "Can't open %s for reading.\n", inName);
        return 1;
    }
    else {
       curcol = 1;

       while ( (c = getc (in)) != EOF ) {
            if ( c == '\n' ) {
               putchar ('\n');
               curcol = 0;
            }
            else if ( curcol >= m  &&  curcol <= n )
               putchar (c);

            ++curcol;
        }
    }
```

```
}
```