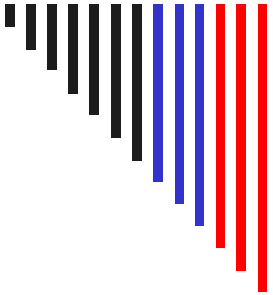


Chapter 14
Bitwise Operators

Objectives

- To be able to use the bitwise logical operators in programs
- To be able to use the bitwise shift operators in programs
- To understand how to create and use masks to manipulate bits

14-1 Exact Size Integer Types

The integer types, such as **int** *and* **long**, *are machine dependent. In one computer, the size of* **int** *may be four bytes; in another computer it may be two bytes. While many bitwise applications work well on machine-dependent integer types, other applications need to assure that the size is fixed. C allows us to define integer types of sizes 8, 16, 32, and 64 bits.*

	Type	Description
Signed	`int8_t`	8-bit signed integer
	`int16_t`	16-bit signed integer
	`int32_t`	32-bit signed integer
	`int64_t`	64-bit signed integer
Unsigned	`uint8_t`	8-bit unsigned integer
	`uint16_t`	16-bit unsigned integer
	`uint32_t`	32-bit unsigned integer
	`uint64_t`	64-bit unsigned integer

Table 14-1 Fixed-size Integer Types

14-2 Logical Bitwise Operators

The logical operators look at data as individual bits to be manipulated. Four operators are provided to manipulate bits: bitwise* and (&), *bitwise inclusive* or (|), *bitwise exclusive* or (^), *and one's complement* (~). *The first three are binary operators; the one's complement is a unary operator.

Topics discussed in this section:

Bitwise *and* Operator
Bitwise Inclusive *or* Operator
Bitwise Exclusive *or* Operator
One's Complement Operator

First Operand Bit	Second Operand Bit	Result
0	0	0
0	1	0
1	0	0
1	1	1

Table 14-2 ***And*** **Truth Table**

PROGRAM 14-1 Simple Bitwise And Demonstration

```
1  /* Demonstrate bitwise AND operator
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <stdint.h>
8
9  int main (void)
10 {
11 // Local Declarations
12    uint16_t  num1 = 0x0257;
13    uint16_t  num2 = 0xA463;
14    uint16_t  res;
15
```

PROGRAM 14-1 Simple Bitwise And Demonstration

```
16  // Statements
17     res = num1 & num2;
18
19     printf ("Input and results in hexadecimal:\n");
20     printf ("num1:   %#06X\n", num1);
21     printf ("num2:   %#06X\n", num2);
22     printf ("result: %#06X\n", res);
23
24     return 0;
25  }  // main
```

```
Results:
Input and results in hexadecimal:
num1:   0X0257
num2:   0XA463
result: 0X0043
```

First Bit	Second Bit	Result
0	0	0
0	1	1
1	0	1
1	1	1

Table 14-3 Inclusive *Or* Truth Table

PROGRAM 14-2 Simple Inclusive *or* Demonstration

```
 1  /* Demonstrate the inclusive OR operator
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6  #include <stdlib.h>
 7  #include <stdint.h>
 8
 9  int main (void)
10  {
11  // Local Declarations
12     uint16_t  num1 = 0x0257;
13     uint16_t  num2 = 0xA463;
14     uint16_t  res;
15
```

PROGRAM 14-2 Simple Inclusive *or* Demonstration

```
16  // Statements
17     res = num1 | num2;
18     printf ("Input and results in hexadecimal:\n");
19     printf ("num1: %#06X\n", num1);
20     printf ("num2: %#06X\n", num2);
21     printf ("res:  %#06X\n", res);
22
23     return 0;
24  }  // main
```

```
Results:
Input and results in hexadecimal:
num1: 0X0257
num2: 0XA463
res:  0XA677
```

First Bit	Second Bit	Result
0	0	0
0	1	1
1	0	1
1	1	0

Table 14-4 **Exclusive *Or* Truth Table**

PROGRAM 14-3 Simple Exclusive *or* Demonstration

```
 1  /* Demonstrate the use of the exclusive or.
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6  #include <stdlib.h>
 7  #include <stdint.h>
 8
 9  int main (void)
10  {
11  // Local Declarations
12     uint16_t  num1 = 0x0257;
13     uint16_t  num2 = 0xA463;
14     uint16_t  res;
15
16  // Statements
17     res = num1 ^ num2;
18
```

PROGRAM 14-3 Simple Exclusive *or* Demonstration

```
19  // Print results in hexadecimal
20     printf ("Input and results in hexadecimal:\n");
21     printf ("num1: %#06X\n", num1);
22     printf ("num2: %#06X\n", num2);
23     printf ("res:  %#06X\n", res);
24
25     return 0;
26  }  // main
```

```
Results:
Input and results in hexadecimal:
num1: 0X0257
num2: 0XA463
res:  0XA634
```

Original Bit	Result
0	1
1	0

Table 14-5 One's Complement Truth Table

PROGRAM 14-4 One's Complement

```
 1  /* Demonstrate use of one's complement
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6  #include <stdlib.h>
 7  #include <stdint.h>
 8
 9  int main (void)
10  {
11  // Local Declarations
12     uint16_t  num = 0x0257;
13     uint16_t  res;
14
15  // Statements
16     res = ~num;
17
```

PROGRAM 14-4 One's Complement

```
18  // Print results in hexadecimal
19     printf ("Input and results in hexadecimal:\n");
20     printf ("num: %#06X\n", num);
21     printf ("res: %#06X\n", res);
22
23     return 0;
24  }  // main
```

```
Results:
Input and results in hexadecimal:
num: 0X0257
res: 0XFDA8
```

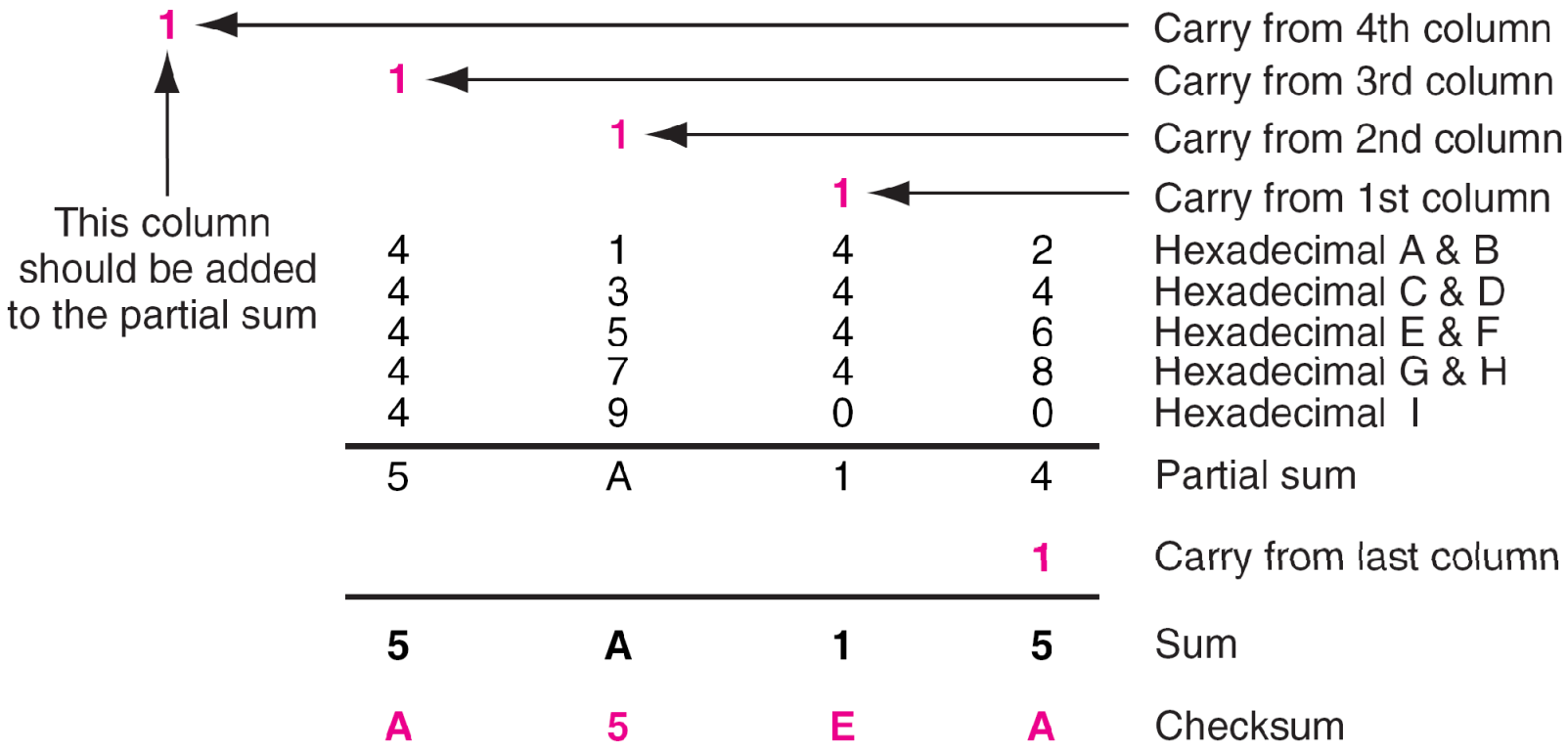


FIGURE 14-1 Checksum Calculation

PROGRAM 14-5 Demonstrate Checksum

```
1  /* Demonstrate the calculation of a checksum using
2     one's complement arithmetic.
3        Written by:
4        Date:
5  */
6  #include <stdio.h>
7  #include <string.h>
8  #include <stdint.h>
9
10 int main (void)
11 {
12 // Local Declarations
13    uint32_t sum      = 0x00000000;
14    uint16_t checksum = 0x0000;
15    char*    str      = "ABCDEFGHI";
16    int      len;
17
18 // Statements
19    len = strlen (str);
20    if (len % 2  == 1)
```

PROGRAM 14-5 Demonstrate Checksum

```
21          // Make the number of characters even
22          len++;
23
24      for (int i = 0; i < len; i += 2)
25          sum = (sum + str[i] * 256 + str[i + 1]);
26
27      // Add carries into lower 16 bits
28      while (sum >> 16)
29          sum = (sum & 0xffff) + (sum >> 16);
30
31      // Complement
32      checksum = ~sum;
33
34      printf ("str:      %s\n",    str);
35      printf ("checksum: %#06X\n", checksum);
36
37      return 0;
38  }   // main
```

```
Results:
str:      ABCDEFGHI
checksum: 0XA5EA
```

14-3 Shift Operators

The shift operators move bits to the right or the left. When applied to unsigned numbers, these operators are implementation independent. When used with signed numbers, however, the implementation is left to the discretion of the software engineer who designs the compiler.

Topics discussed in this section:

Shift

Rotation

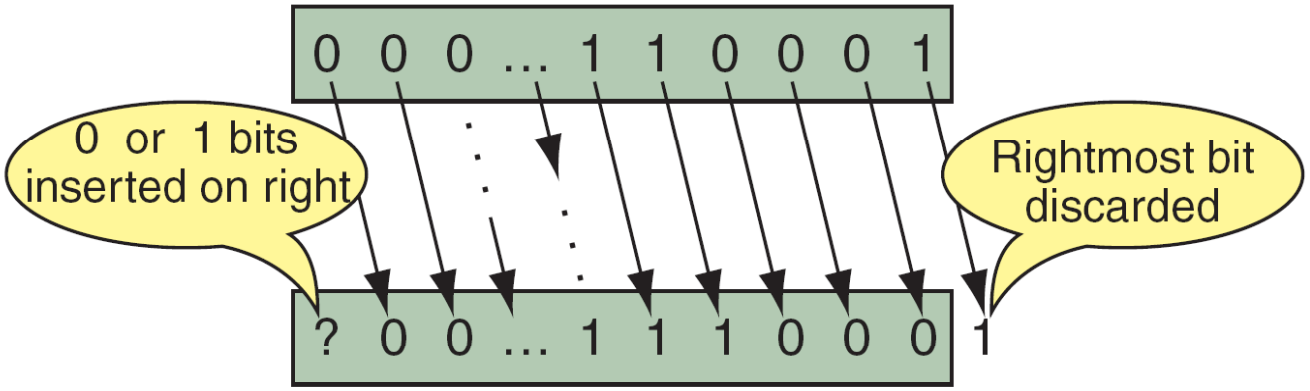


FIGURE 14-2 Shift-right Operation

PROGRAM 14-6 Simple Shift-right Demonstration

```
1  /* Demonstrate the bitwise shift-right operator.
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <stdint.h>
8
9  // Function Declaration
10 void bin16 (uint16_t num, char* bitStr);
11
12 int main (void)
13 {
14 // Local Definitions
15    uint16_t num  = 0x0040;
16    uint16_t res;
17    char     bitStr[17] = {0};
18
```

PROGRAM 14-6 Simple Shift-right Demonstration

```
19  // Statements
20     bin16 (num, bitStr);
21     printf("Original value:  %s (%#06x)\n",
22            bitStr, num);
23
24     res =  num >> 1;
25     bin16 (res, bitStr);
26     printf("Shifted 1 right: %s (%#06x)\n",
27            bitStr, res);
28
29     res =  num >> 2;
30     bin16 (res, bitStr);
31     printf("Shifted 2 right: %s (%#06x)\n",
32            bitStr, res);
33
34     res = num >> 4;
35     bin16 (res, bitStr);
36     printf("Shifted 4 right: %s (%#06x)\n",
37            bitStr, res);
38
39     return 0;
40  }  // main
41
```

PROGRAM 14-6 Simple Shift-right Demonstration

```
42  /* ========================= bin16 ========================
43     Convert fixed 16-bit integer to binary digit string.
44        Pre  num contains integral value to be converted
45             bitStr is pointer to variable for bit string
46        Post bit string stored in str
47  */
48  void bin16 (uint16_t num, char* bitStr)
49  {
50  // Statements
51     for (int i = 0; i < 16; i++)
52          bitStr[i] = (char) ((num >> 15 - i) & 0X0001) +
    48;
53     return;
54   } // bin16
```

```
Results:
Original value:  0000000001000000 (0x0040)
Shifted 1 right: 0000000000100000 (0x0020)
Shifted 2 right: 0000000000010000 (0x0010)
Shifted 4 right: 0000000000000100 (0x0004)
```

$2^{\text{shift value}}$	Divides by	Shift Operator
1	2	>> 1
2	4	>> 2
3	8	>> 3
4	16	>> 4
...	...	...
n	2^n	>> *n*

Table 14-6 Divide by Shift

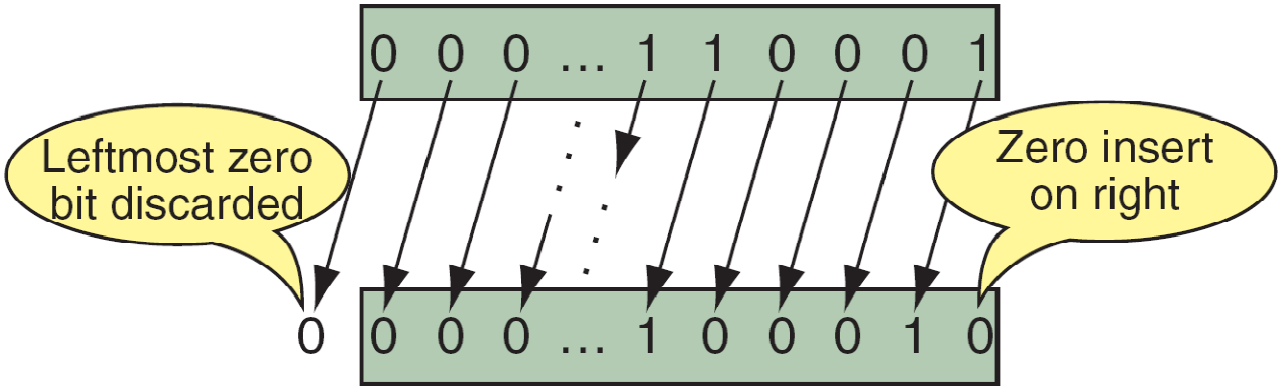


FIGURE 14-3 Shift-left Operation

PROGRAM 14-7 Simple Shift-left Demonstration

```
 1  /* Demonstrate the bitwise shift-left operator.
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6  #include <stdlib.h>
 7  #include <stdint.h>
 8
 9  #include "bin16.c"
10
11  int main (void)
12  {
13  // Local Definitions
14     uint16_t num  = 0x0031;
15     uint16_t res;
16     char     bitStr[17] = {"0"};
17
18  // Statements
19     bin16 (num, bitStr);
20     printf("Original value: %s (%#06x)\n", bitStr, num);
21
```

PROGRAM 14-7 Simple Shift-left Demonstration

```
22      res = num << 1;
23      bin16 (res, bitStr);
24      printf("Shifted 1 left: %s (%#06x)\n", bitStr, res);
25
26      res =  num << 2;
27      bin16 (res, bitStr);
28      printf("Shifted 2 left: %s (%#06x)\n", bitStr, res);
29
30      res = num << 4;
31      bin16 (res, bitStr);
32      printf("Shifted 4 left: %s (%#06x)\n", bitStr, res);
33
34      return 0;
35  }   // main
```

```
Results:
Original value: 0000000000110001 (0x0031)
Shifted 1 left: 0000000001100010 (0x0062)
Shifted 2 left: 0000000011000100 (0x00c4)
Shifted 4 left: 0000001100010000 (0x0310)
```

$2^{\text{shift value}}$	Multiplies by	Shift Operator
1	2	<< 1
2	4	<< 2
3	8	<< 3
4	16	<< 4
...	...	...
n	2^n	<< *n*

Table 14-7 Multiply by Shift

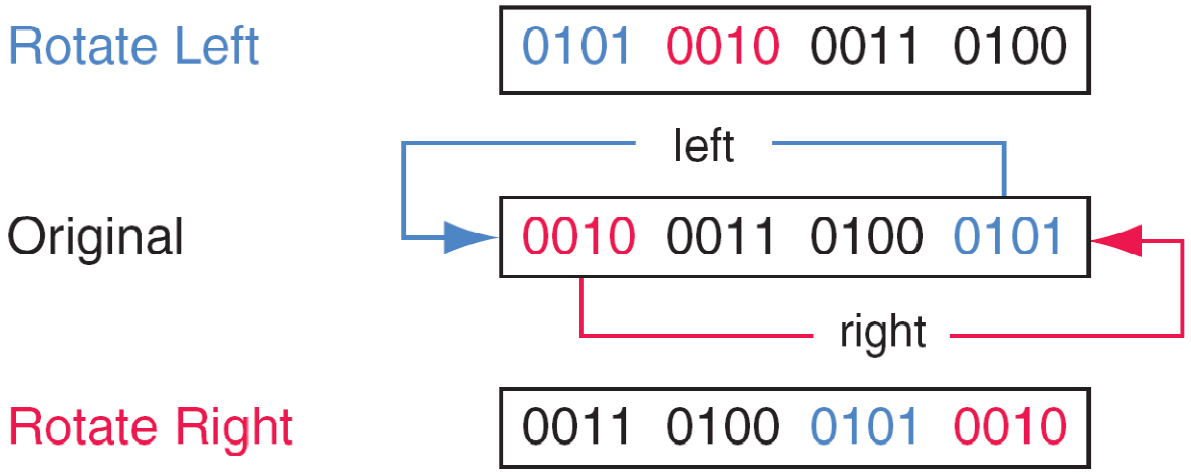


FIGURE 14-4 Right and Left Rotation

PROGRAM 14-8 Rotate Left and Right Test Driver

```
1  /* Test driver for rotate left and right.
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6  #include <stdint.h>
7
8  // Function Declarations
9  uint16_t rotate16Left  (uint16_t num , int n);
10 uint16_t rotate16Right (uint16_t num , int n);
11
12 int main (void)
13 {
14 // Local Declaration
15    uint16_t num   = 0X2345;
16
```

PROGRAM 14-8 Rotate Left and Right Test Driver

```
17  // Statements
18     printf("Original:       %#06X\n", num);
19     printf("Rotated Left:   %#06X\n",
20            rotate16Left (num, 4));
21     printf("Rotated Right: %#06X\n",
22            rotate16Right(num, 4));
23  } // main
24
25  /* ================== rotate16Left ==================
26     Rotate 16 bit fixed size integer left n bits.
27        Pre  num is a fixed size 16-bit integer
28        Post num rotated n bits left
29  */
30  uint16_t rotate16Right (uint16_t num , int n)
31  {
32     return  ( ( num  << n ) | ( num >> 16 - n ) );
```

PROGRAM 14-8 Rotate Left and Right Test Driver

```
33  }  // rotate16Right
34
35  /* ==================== rotate16Right ==================
36     Rotate 16 bit fixed size integer right n bits.
37        Pre  num is a fixed size 16-bit integer
38        Post num rotated n bits left
39  */
40  uint16_t rotate16Left (uint16_t num , int n)
41  {
42     return  ( ( num >> n ) | ( num << 16 - n ) );
43  }  // rotate16Left
```

```
Results:
Original:       0X2345
Rotated Left:   0X5234
Rotated Right:  0X3452
```

14-4 Masks

In many programs, bits are used as binary flags: 0 is off, and 1 is on. To set and test the flags, we use a bit mask. A mask is a variable or constant, usually stored in a byte or short integer. The bits are numbered from the least significant bit (rightmost), starting at 0.

Topics discussed in this section:

Creating Masks
Using Masks

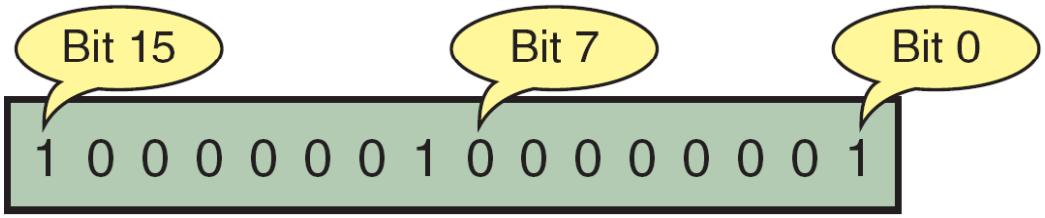


FIGURE 14-5 **Bit Mask in a 16-bit Integer**

PROGRAM 14-9 Determine Network Address

```
1  /* Given a host address and the size of the prefix,
2     determine its network address.
3          Written by:
4          Date:
5  */
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include <stdint.h>
9
10 int main (void)
11 {
12 // Local Declarations
13    unsigned int comAddr[4];
14    unsigned int mask[4];
15    unsigned int netAddr[4];
16    uint32_t     comAd   = 0;
17    uint32_t     mask32  = 0;
18    uint32_t     netAd   = 0;
19    int          prefix;
20
```

PROGRAM 14-9 Determine Network Address

```
21  // Statements
22     printf ("Enter host address <x.y.z.t>: ");
23     scanf ("%d%*c%d%*c%d%*c%d",
24           &comAddr[3], &comAddr[2], &comAddr[1],
25           &comAddr[0]);
26
27     printf ("Enter prefix: ");
28     scanf ("%d", &prefix);
29
30     // Convert address to a 32-bit computer address
31     for (int i = 3; i >= 0; i--)
32          comAd = comAd * 256 + comAddr[i];
33
34     // Create a 32-bit mask
35     for (int i = 32 - prefix; i < 32; i++)
36          mask32 = mask32 | (1 << i);
37
38     // AND to get a 32-bit Network Address
39     netAd = comAd & mask32;
40
```

PROGRAM 14-9 Determine Network Address

```
41      // Change mask into the form x.y.z.t
42      for (int i = 0; i < 4; i++)
43          {
44           mask[i] = mask32 % 256;
45           mask32  = mask32 / 256;
46          } // for
47
48      // Change IP address into the form x.y.z.t
49      for (int i = 0; i < 4; i++)
50          {
51           netAddr[i] = netAd % 256;
52           netAd      = netAd / 256;
53          } // for
54
55      // Print Addresses
56      printf ("\nAddresses:\n");
57      printf ("Computer Address: ");
58      printf ("%d.%d.%d.%d\n",
59               comAddr[3], comAddr[2],
60               comAddr[1], comAddr[0]);
```

PROGRAM 14-9 Determine Network Address

```
61
62      printf ("Mask:                ");
63      printf ("%d.%d.%d.%d\n",
64               mask[3], mask[2], mask[1], mask[0]);
65
66      printf ("Net Address:         ");
67      printf ("%d.%d.%d.%d\n",
68               netAddr[3], netAddr[2], netAddr[1],
69               netAddr[0]);
70      return 0;
71  }   // main
```

```
Results:
Enter host address <x.y.z.t>: 123.45.78.12
Enter prefix: 18

Addresses:
Computer Address: 123.45.78.12
Mask:             255.255.192.0
Net Address:      123.45.64.0
```

PROGRAM 14-10 Determine Last Address in a Network

```
1  /* Determine the last address in a broadcast network.
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6  #include <math.h>
7  #include <stdint.h>
8
9  int main (void)
10 {
11 // Local Declarations
12    unsigned int  comAddr[4];
13    unsigned int  mask[4];
14    unsigned int  broadAddr[4];
15
16    uint32_t comAd   = 0;
17    uint32_t mask32  = 0;
18    uint32_t broadAd = 0;
19    int      prefix;
20
```

PROGRAM 14-10 Determine Last Address in a Network

```
21  // Statements
22     printf ("Enter host address <x.y.z.t>: ");
23     scanf ("%d%*c%d%*c%d%*c%d",
24             &comAddr[3], &comAddr[2],
25             &comAddr[1], &comAddr[0]);
26     printf("Enter prefix: ");
27     scanf ("%d", &prefix);
28
29     // Convert address to 32-bit computer address
30     for (int i = 3; i >= 0; i--)
31          comAd = comAd * 256 + comAddr[i];
32
33     // Create 32-bit prefix mask
34     for (int i = 32-prefix; i < 32; i++)
35          mask32 = mask32 | (0x0001 << i);
36
37     // And to get a 32-bit Network Address
38     broadAd = comAd | (~mask32);
39
```

PROGRAM 14-10 Determine Last Address in a Network

```
40      // Change the mask into the form x.y.z.t
41      for (int i= 0; i < 4; i++)
42           {
43            mask[i]= mask32 % 256;
44            mask32 = mask32 / 256;
45           } // for
46
47      // Change the IP address to the form x.y.z.t
48      for (int i= 0; i < 4; i++)
49           {
50            broadAddr[i] = broadAd % 256;
51            broadAd      = broadAd / 256;
52           } // for
53
54      printf ("\nPrinting Addresses\n");
55      printf ("Computer Address:  ");
56      printf ("%d.%d.%d.%d\n",
57               comAddr[3], comAddr[2],
    comAddr[1],comAddr[0]);
58      printf ("Mask:              ");
59      printf ("%d.%d.%d.%d\n",
60               mask[3], mask[2], mask[1], mask[0]);
```

PROGRAM 14-10 Determine Last Address in a Network

```
61
62      printf ("Broadcast Address: ");
63      printf ("%d.%d.%d.%d\n",
64               broadAddr[3], broadAddr[2],
65               broadAddr[1], broadAddr[0]);
66      return 0;
67  }   // main
```

```
Results:
Enter host address <x.y.z.t>: 123.45.78.12
Enter prefix: 18

Printing Addresses
Computer Address:  123.45.78.12
Mask:              255.255.192.0
Last Address:      123.45.127.255
```

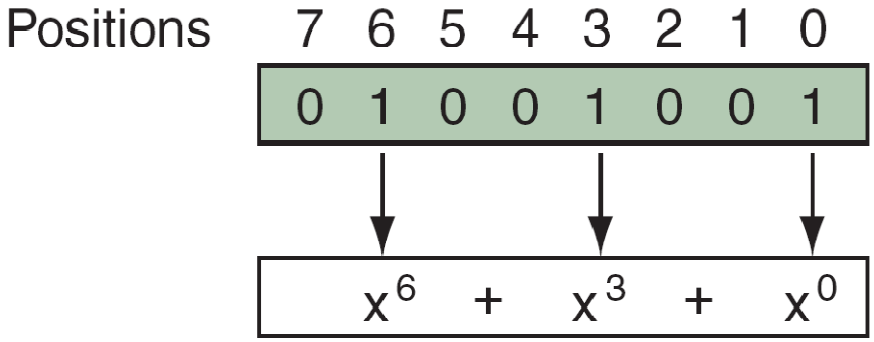
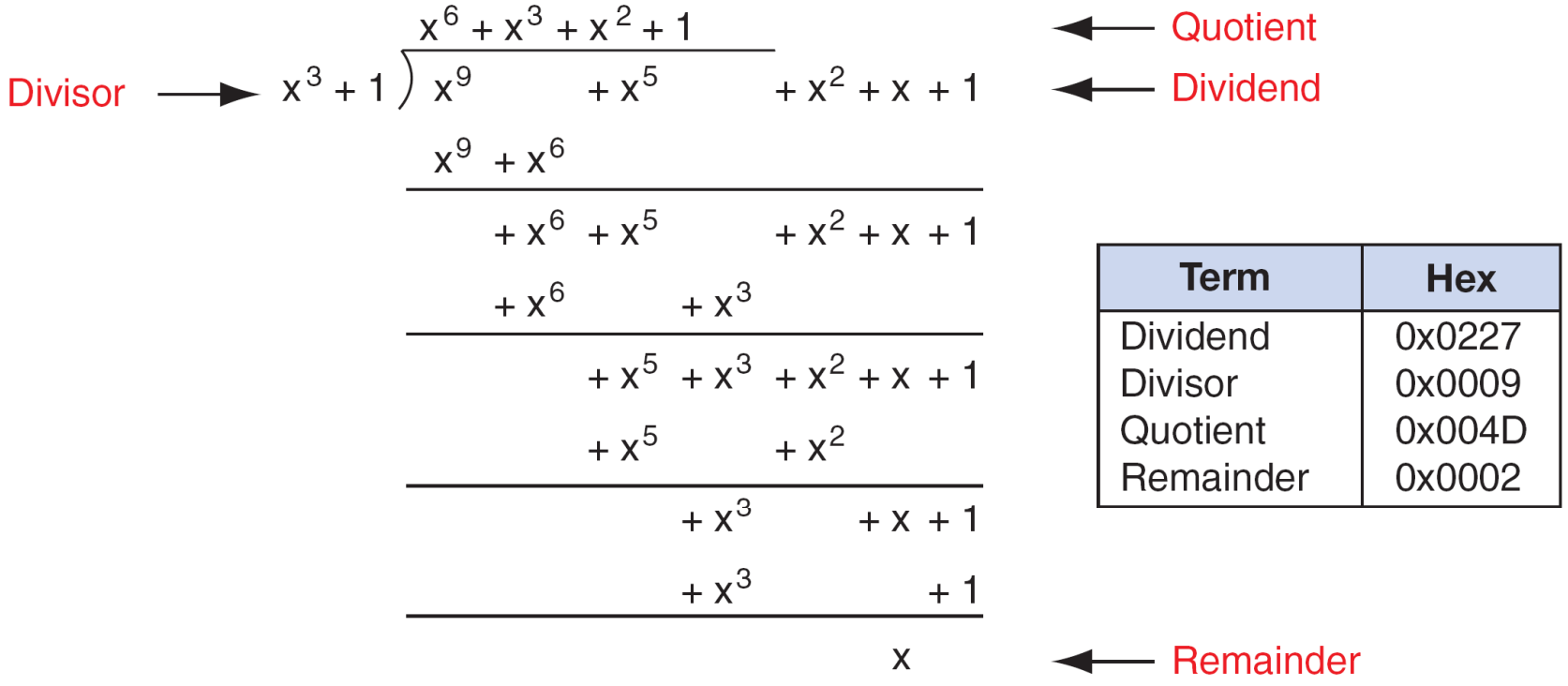


FIGURE 14-6 Polynomial Coefficients



Term	Hex
Dividend	0x0227
Divisor	0x0009
Quotient	0x004D
Remainder	0x0002

FIGURE 14-7 Polynomial Division

PROGRAM 14-11 Polynomial Division

```
 1  /* Demonstrate polynomial division.
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6  #include <stdint.h>
 7
 8  // Function Declaration
 9  int degree (uint16_t);
10
11  int main (void)
12  {
13  // Local Declaration
14     uint16_t dvdn = 0X0227;
15     uint16_t dvsr = 0X0009;
16     uint16_t qtnt = 0X0000;
17     uint16_t rmdr;
18     uint16_t q;
19     int      dgre;
20
```

PROGRAM 14-11 Polynomial Division

```
21  // Statements
22     printf ("Dividend:  %#06X\n", dvdn);
23     printf ("Divisor:   %#06X\n", dvsr);
24
25     rmdr = dvdn;
26     while ((dgre = degree (dvdn) - degree (dvsr)) >= 0)
27        {
28         q    = 1 << dgre;
29         rmdr = dvdn ^ (dvsr << degree (q));
30         qtnt = qtnt | q;
31         dvdn = rmdr;
32        } // while
33
34     printf ("Quotient : %#06X\n", qtnt);
35     printf ("Remainder: %#06X\n", rmdr);
36     return 0;
37  }  // main
38
39  /* Determine degree of polynomial represented by
```

PROGRAM 14-11 Polynomial Division

```
40      a fixed-length 16-bit variable.
41          Pre  poly represents a polynomial
42          Post degree returned
43  */
44  int degree (uint16_t  poly)
45  {
46  // Local Declarations
47      uint16_t mask = 0X0001;
48      uint16_t temp;
49      int      pos= -1;
50
51  // Statements
52       for (int i = 0; i <16; i++)
53           {
54            temp = poly >> i;
55            if ((temp & mask) == 1)
56                 pos = i;
57           } // for
58      return pos;
59  }  // degree
```

PROGRAM 14-11 Polynomial Division

```
Results:
Dividend:  0X0227
Divisor:   0X0009
Quotient :  0X004D
Remainder: 0X0002
```

14-5 Software Engineering

In Chapter 12, we looked at what makes a good function. In this section, we look at how you design good programs.

Topics discussed in this section:

Payroll Case Study
Structure Chart Design

Payroll Case Study

1. Requirements:

   Given employees and their hours worked, compute net pay and record all payroll data for subsequent processing, such as W2 statements. Prepare paychecks and a payroll ledger.

   Maintain data on a sequential payroll file.

2. Provide for the following nonstatutory deductions:

   a. Health plan
   b. United Way
   c. Union dues

3. The payroll data are:

   a. Employee number
   b. Pay rate
   c. Union member flag
   d. United Way contribution
   e. Exemptions

FIGURE 14-8 Requirements for Payroll Case Study

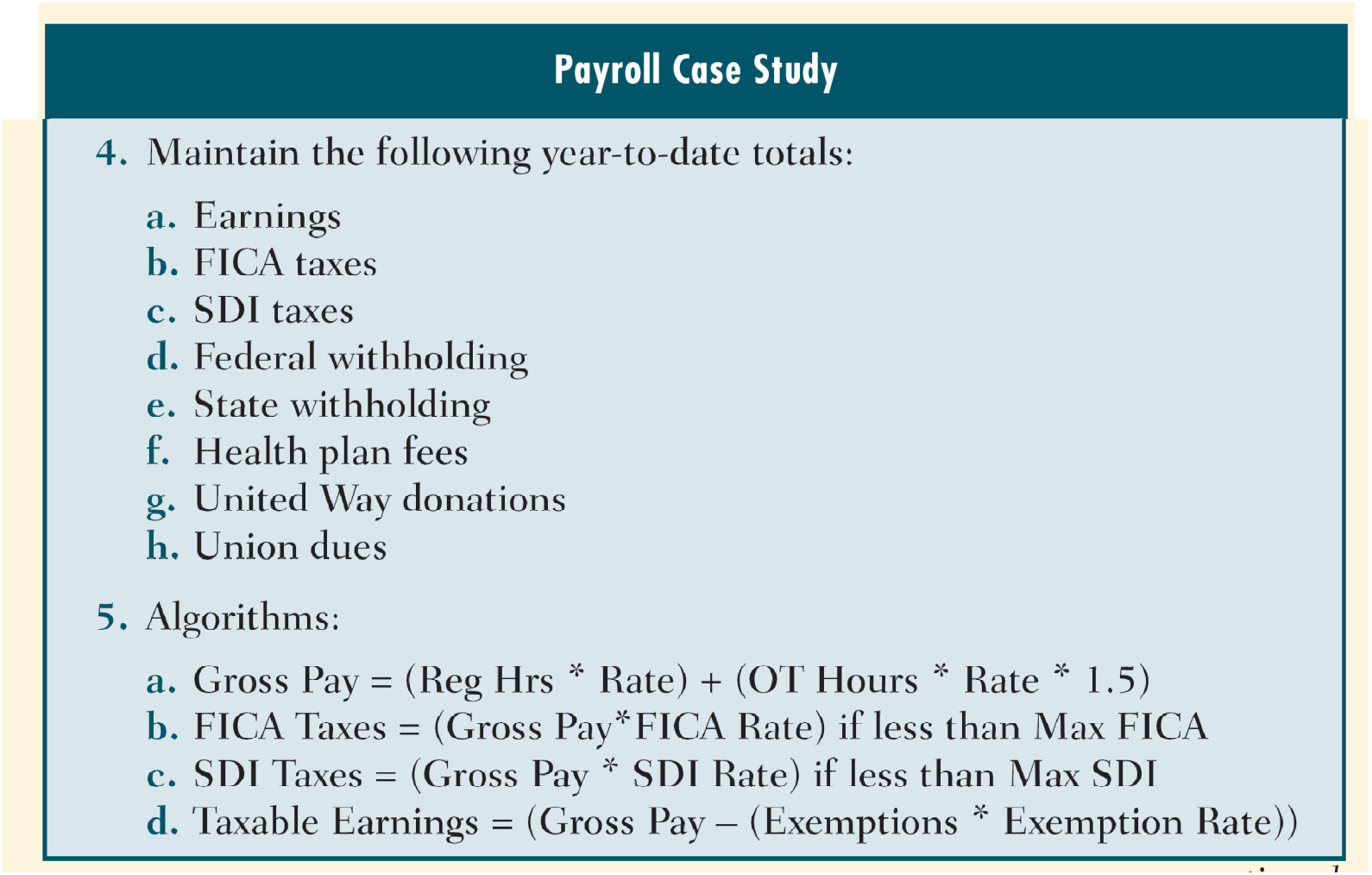


FIGURE 14-8 Requirements for Payroll Case Study (*continued*)

Payroll Case Study

e. Federal Taxes = (Taxable Earnings * Federal TaxRate)
f. State Taxes = (Taxable Earnings * State Tax Rate)
g. Net Pay = (Gross Pay – (FICA Taxes + SDI Taxes +
Federal Taxes + State Taxes + Health Fee +
United Way Donation + Union Dues))

FIGURE 14-8 Requirements for Payroll Case Study (*continued*)

Note

Good programs start with a good structure chart design.

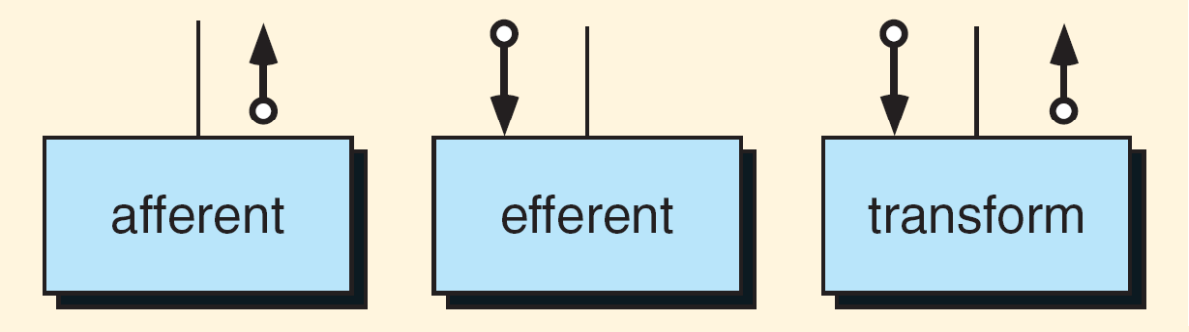


FIGURE 14-9 Afferent, Efferent, and Transform Modules

Module	Afferent	Efferent	Transform
Read hours worked	✓		
Compute pay			✓
Maintain payroll master file	✓	✓	
Prepare paychecks		✓	
Prepare payroll ledger		✓	
Calculate nonstatutory deductions			✓
Calculate year-to-date totals			✓
Calculate gross pay			✓
Calculate taxes			✓

Table 14-8 Classification of Payroll Modules

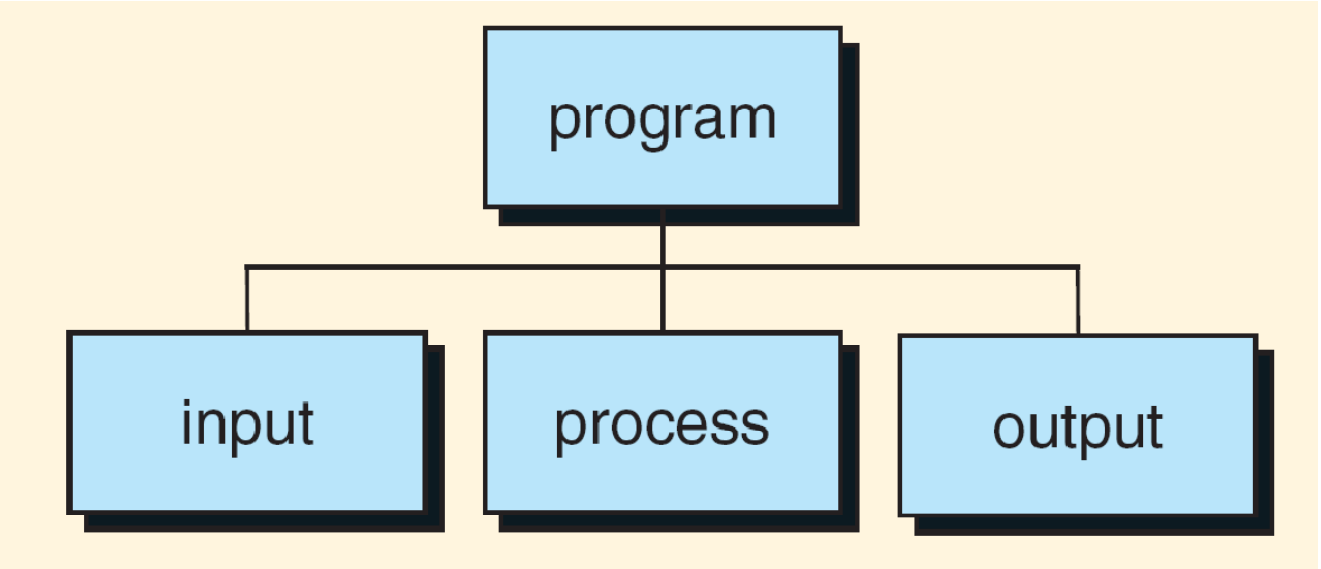


FIGURE 14-10 Streams