Solutions to Chapter 12

Review Questions

1. b. False
3. b. False
5. a. True
7. d. type definition
9. d. The enumerated values are automatically assigned constant values unless otherwise directed.
11. C normally uses two different structure declarations: tagged and type-defined.
13. c. `stu.major`
15. d. union
17.
    a. true
    b. false
    c. false
    d. true

Exercises

19.

```
typedef struct
   {
    int   partNo;
    char* descr;
    int   reOrder;
    int   onHand;
    char  unitMeas[9];
    float price;
   } ITEM;
```

21.

```
struct ARRAY_ELEMENT
   {
    char* month;
    int   days;
    char* activity;
   };

   struct ARRAY_ELEMENT calendar [366];
```

23.
    a. valid
    b. not valid: `y` is pointer to a character
    c. valid
    d. valid
    e. valid

25.

```
12.450000
66
23.340000
```

Problems

27. See Program 12-1.

Program 12-1 Solution to Problem 27

```
// Global Declaration
   typedef struct
      {
       char* month;
       int   day;
       int   year;
      } DATE;

   typedef struct
      {
       int   month;
       int   days;
       char* name;
      } MONTH;

/* ===================== increment ====================
   This function increments date held in a DATE
   structure by one day.
      Pre   oldDate is a DATE structure
      Post  returns a structure containing the new date.
*/
DATE increment (DATE oldDate)
{
// Local Definitions
   int   i;
   DATE  newDate;
   MONTH table[12] = {{1, 31, "JAN"},
                      {2, 28, "FEB"},
                      {3, 31, "MAR"},
                      {4, 30, "APR"},
                      {5, 31, "MAY"},
                      {6, 30, "JUN"},
                      {7, 31, "JUL"},
                      {8, 31, "AUG"},
                      {9, 30, "SEP"},
                      {10,31, "OCT"},
                      {11,30, "NOV"},
                      {12,31, "DEC"}};
// Statements
   // check for leap year
   if (!(oldDate.year % 4)
     && (oldDate.year % 100)
     || !(oldDate.year % 400))
         table[1].days++;

   newDate = oldDate;
   newDate.day++;

// find the name of month from table
   i = 0;
   while (strcmp (newDate.month, table[i].name))
      i++;

   if (newDate.day > table[i].days)
      {
       newDate.day = 1;
```

Program 12-1 Solution to Problem 27 (continued)

```
        if (!strcmp (newDate.month, table[11].name))
           {
            newDate.year++;
            newDate.month = table[0].name;
           } // if next year
        else
            newDate.month = table[i + 1].name;
      } // if next month
   return newDate;
}  // increment
```

29. See Program 12-2.

Program 12-2 Solution to Problem 29

```
/* ======================== later =====================
   This function returns true if first date is
   earlier than second date and false if first date
   is later.
      Pre   a structure containing the first date
            a structure containing the second date
            (assumes different dates)
      Post  returns boolean
*/
bool later (DATE d1, DATE d2)
{
// Local Declarations
   int    d1Index;
   int    d2Index;
   bool   retval;
   MONTH  table[12] =
      {
       {1, 31, "JAN"},
       {2, 28, "FEB"},
       {3, 31, "MAR"},
       {4, 30, "APR"},
       {5, 31, "MAY"},
       {6, 30, "JUN"},
       {7, 31, "JUL"},
       {8, 31, "AUG"},
       {9, 30, "SEP"},
       {10,31, "OCT"},
       {11,30, "NOV"},
       {12,31, "DEC"}
   }; // MONTH table

// Statements
   // check the year
   if (d1.year < d2.year)
       retval = true;
   else if (d1.year > d2.year)
       retval = false;
   else // same year
      {
       d1Index = 0;
       while (strcmp (d1.month, table[d1Index].name))
          d1Index++;
       d2Index = 0;
       while (strcmp (d2.month, table[d2Index].name))
          d2Index++;

       if (d1Index < d2Index)
           retval = true;
       else if (d1Index > d2Index)
           retval = false;
       else // same month
```

Program 12-2 Solution to Problem 29 (continued)

```
          {
           if (d1.day < d2.day)
               retval = true;
           else if (d1.day > d2.day)
               retval = false;
           else // same date
               retval = false;
          } // else same month
      } // else same year
   return retval;
}  // later
```

31. See Program 12-3.

Program 12-3 Solution to Problem 31

```
#define TOLERANCE .00001

// Global Declaration
   typedef struct
      {
       int numerator;
       int denominator;
      } FRACTION;

/* ==================== fractionCmp ==================
   This function compares two fraction structures.
   If fractions are equal, it returns zero.
   If first parameter <  fraction in second parameter,
   it returns a negative number.  Otherwise,
   it returns a positive number.
      Pre   two fraction structures
      Post  returns the result of comparison
*/
int fractionCmp (FRACTION fr1, FRACTION fr2)
{
// Local Declarations
   int   retval;
   float a;
   float b;

// Statements
   a = (float) fr1.numerator / fr1.denominator;
   b = (float) fr2.numerator / fr2.denominator;

   if (fabs (a - b) < TOLERANCE)
       retval = 0;
   else if (a < b)
       retval = -1;
   else
       retval = 1;
   return retval;
}  // fractionCmp
```

33. See Program 12-4.

Program 12-4 Solution to Problem 33

```
// Global Declaration
   typedef struct
      {
       int x;
       int y;
      } POINT;
```

Program 12-4 Solution to Problem 33 (continued)

```
   typedef struct
      {
       POINT beg;
       POINT end;
      } LINE;

/* ======================== getLine =====================
   This function accepts two POINTs and returns a
   structure representing a LINE.
      Pre   the two structure representing the points
      Post  returns structure representing a LINE
*/
LINE getLine (POINT p1, POINT p2)
{
// Local Declarations
   LINE line;

// Statements
   line.beg.x = p1.x;
   line.beg.y = p1.y;
   line.end.x = p2.x;
   line.end.y = p2.y;

   return line;
}  // getLine
```

35. See Program 12-5.

Program 12-5 Solution to Problem 35

```
// Global Declarations
   typedef struct
      {
       char* suit;
       int   value;
      } CARD;

   typedef CARD DECK[52];

/* ======================== shuffle =====================
   This function  shuffles a deck of cards.
      Pre   A deck of cards that has been initialized
      Post  The deck has been shuffled
*/
void shuffle (DECK deck)
{
// Local Declarations
   int  random;
   CARD temp;

// Statements
   srand (time (NULL));

   for (int cur = 0; cur < 52; cur++)
      {
       random       = rand() % 52;
       temp         = deck[random];
       deck[random] = deck[cur];
       deck [cur]   = temp;
      } // for

   return;
}  // shuffle
```

37. See Program 12-6.

Program 12-6 Solution to Problem 37

```
/* This program creates and prints a sorted name list.
      Written by:
      Date:
*/

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

#define ARRAY_SIZE 5
#define FIRST_SIZE 20
#define LAST_SIZE 30
#define COMPANY_SIZE 40
#define FLUSH while(getchar() != '\n')

// Global Declarations
   typedef struct
      {
       char first[FIRST_SIZE];
       char init;
       char last[LAST_SIZE];
      } PERSON;

   typedef struct
      {
       char type;
       union
          {
           char   company[COMPANY_SIZE];
           PERSON person;
          } un;
      } NAME;

// Function Declarations
   void buildArray (NAME ary[]);
   void sortArray  (NAME ary[]);
   void copy       (NAME *dest, NAME src);
   int  lessThan   (NAME name1, NAME name2);
   void printArray (NAME ary[]);

int main (void)
{
// Local Declarations
   NAME ary[ARRAY_SIZE];

// Statements
   buildArray (ary);
   sortArray  (ary);
   printArray (ary);

   return 0;
}  // main

/* ==================== buildArray ====================
   This function fills an array with NAMEs
      Pre   ary is an array of NAME structures
      Post  Array has been filled with names
*/
const char* errMsg
          = "\n\a**Invalid code - try again \n\n";

void buildArray (NAME ary[])
{
// Local Declarations
   char input;
   int  index = 0;
```

Program 12-6 Solution to Problem 37 (continued)

```
// Statements
   while (index < ARRAY_SIZE)
      {
       printf ("Enter C for company--P for person: ");
       scanf  ("%c", &input);
       FLUSH;
       input = toupper(input);
       switch (input)
          {
           case 'C' :
                ary[index].type = input;
                printf ("\nEnter company name:  ");
                fgets  (ary[index].un.company,
                        COMPANY_SIZE - 1, stdin);
                ary[index].un.company
                        [strlen(ary[index].un.company)-1]
                    = '\0';
                index++;
                break;
           case 'P' :
                ary[index].type = input;
                printf ("\nEnter person name: ");
                scanf  ("%s %c %s",
                          ary[index].un.person.first,
                         &ary[index].un.person.init,
                          ary[index].un.person.last);
                FLUSH;
                index++;
                break;
           default  :
                printf ("%s", errMsg);
                break;
          } // switch Company or Person
      } // for each NAME in ary

   return;
}  // buildArray

/* ================== sortArray ==================
   This function sorts an array of NAME structures
      Pre   ary is an array of NAME structures
      Post  Array has been sorted
*/
void sortArray  (NAME ary[])
{
// Local Declarations
   int  walker1;
   int  walker2;
   NAME temp;

// Statements
   for (walker1 = 1; walker1 < ARRAY_SIZE; walker1++)
      {
       walker2 = walker1;
       copy (&temp, ary[walker2]);
       while (walker2 > 0
          &&  lessThan (temp, ary[walker2 - 1]))
          {
           copy (&ary[walker2], ary[walker2 - 1]);
           walker2--;
          } // while
       copy (&ary[walker2], temp);
      } // for
   return;
}  // sortArray

/* ================== copy ====================
```

Program 12-6 Solution to Problem 37 (continued)

```
   This function copies one NAME structure to another
      Pre   src and dest are NAME structures
      Post  src copied to dest
*/
void copy (NAME *dest, NAME src)
{
// Statements
   dest->type  =  src.type;
   switch (src.type)
     {
      case 'C' :
           strcpy (dest->un.company, src.un.company);
           break;
      case 'P' :
           strcpy (dest->un.person.first,
                   src.un.person.first);
           dest->un.person.init = src.un.person.init;
           strcpy (dest->un.person.last,
                   src.un.person.last);
           break;
     } // switch Company or Person
   return;
}  // copy

/* ================== lessThan ====================
   This function compares 2 NAME structures
      Pre   name1 and name2 are NAME structures
      Post  NAME structures compared
            Returns 1 if name1 < name2, 0 otherwise
*/
int lessThan (NAME name1, NAME name2)
{
// Local Declarations
   int   length;
   int   retval;
   char  ary1[60];
   char  ary2[60];
   char* walker1;
   char* walker2;

// Statements
   switch (name1.type)
      {
       case 'C' : walker1 = name1.un.company;
                  walker2 = ary1;
                  while (*walker1 != '\0')
                     {
                      if (!isspace (*walker1))
                         {
                          *walker2 = toupper(*walker1);
                          walker2++;
                         } // if
                      walker1++;
                     } // while
                  *walker2 = '\0';
                  break;
       case 'P' : strcpy (ary1, name1.un.person.last);
                  strcat (ary1, name1.un.person.first);
                  length = strlen (ary1);
                  ary1[length] =  name1.un.person.init;
                  ary1[length + 1] = '\0';
                  walker1          = ary1;
                  while (*walker1 != '\0')
                     {
                      *walker1 = toupper(*walker1);
                      walker1++;
                     } // while
                  break;
```

Program 12-6 Solution to Problem 37 (continued)

```
      } // switch Company or Person

   switch (name2.type)
      {
       case 'C' : walker1 = name2.un.company;
                  walker2 = ary2;
                  while (*walker1 != '\0')
                     {
                      if (!isspace (*walker1))
                         {
                          *walker2 = toupper(*walker1);
                          walker2++;
                         } // if
                      walker1++;
                     } // while
                  *walker2 = '\0';
                  break;
       case 'P' : strcpy (ary2, name2.un.person.last);
                  strcat (ary2, name2.un.person.first);
                  length = strlen (ary2);
                  ary2[length] =  name2.un.person.init;
                  ary2[length + 1] = '\0';
                  walker1          = ary2;
                  while (*walker1 != '\0')
                     {
                      *walker1 = toupper(*walker1);
                      walker1++;
                     } // while
                  break;
      } // switch Company or Person

   if (strcmp (ary1, ary2)  <  0)
      retval = 1;
   else
      retval = 0;

   return retval;
}  // lessThan

/* =================== printArray ===================
   This function prints an array of NAME structures
      Pre   ary is an array of NAME structures
      Post  Array has been printed
*/

void printArray (NAME ary[])
{
// Statements
   for (int index = 0; index < ARRAY_SIZE; index++)
      {
       switch (ary[index].type)
          {
           case 'C' :
                printf ("Company: %s\n",
                          ary[index].un.company);
                break;
           case 'P' :
                printf ("Person : %s %c. %s\n",
                          ary[index].un.person.first,
                          ary[index].un.person.init,
                          ary[index].un.person.last);
                break;
          } // switch Company or Person
      } // for each NAME
   return;
}  // printArray
```