1. （洗牌和發牌）修改圖 7.24 的程式，使發牌函式能夠一次發出 5 張牌。然後寫出以下的函式：

   a) 判斷這 5 張牌裡是否有對子。

   b) 判斷這 5 張牌裡是否有雙對子。

   c) 判斷這 5 張牌裡是否有三條（如 3 張傑克）。

   d) 判斷這 5 張牌裡是否有鐵支（如 4 張 A）。

   e) 判斷這 5 張牌裡是否有同花（即 5 張同樣花色的牌）。

   f) 判斷這 5 張牌裡是否有順子（即 5 張連續的牌）。

```c
// Fig. 7.24: fig07_24.c
// Card shuffling and dealing.
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define SUITS 4
#define FACES 13
#define CARDS 52

// prototypes
void shuffle( unsigned int wDeck[][ FACES ] ); // shuffling modifies wDeck
void deal( unsigned int wDeck[][ FACES ], const char *wFace[],
   const char *wSuit[] ); // dealing doesn't modify the arrays

int main( void )
{
```

```
18      // initialize suit array
19      const char *suit[ SUITS ] =
20         { "Hearts", "Diamonds", "Clubs", "Spades" };
21
22      // initialize face array
23      const char *face[ FACES ] =
24         { "Ace", "Deuce", "Three", "Four",
25           "Five", "Six", "Seven", "Eight",
26           "Nine", "Ten", "Jack", "Queen", "King" };
27
28      // initialize deck array
29      unsigned int deck[ SUITS ][ FACES ] = { 0 };
30
31      srand( time( NULL ) ); // seed random-number generator
32
33      shuffle( deck ); // shuffle the deck
34      deal( deck, face, suit ); // deal the deck
35   } // end main
36
37   // shuffle cards in deck
38   void shuffle( unsigned int wDeck[][ FACES ] )
39   {
40      size_t row; // row number
41      size_t column; // column number
42      size_t card; // counter
43
44     // for each of the cards, choose slot of deck randomly
45     for ( card = 1; card <= CARDS; ++card ) {
46
47        // choose new random location until unoccupied slot found
48        do {
49           row = rand() % SUITS;
50           column = rand() % FACES;
51        } while( wDeck[ row ][ column ] != 0 ); // end do...while
52
53         // place card number in chosen slot of deck
54         wDeck[ row ][ column ] = card;
55      } // end for
56   } // end function shuffle
57
58   // deal cards in deck
59   void deal( unsigned int wDeck[][ FACES ], const char *wFace[],
60      const char *wSuit[] )
61   {
62      size_t card; // card counter
63      size_t row; // row counter
64      size_t column; // column counter
65
```

```
66      // deal each of the cards
67      for ( card = 1; card <= CARDS; ++card ) {
68         // loop through rows of wDeck
69         for ( row = 0; row < SUITS; ++row ) {
70            // loop through columns of wDeck for current row
71            for ( column = 0; column < FACES; ++column ) {
72               // if slot contains current card, display card
73               if ( wDeck[ row ][ column ] == card ) {
74                  printf( "%5s of %-8s%c", wFace[ column ], wSuit[ row ],
75                     card % 2 == 0 ? '\n' : '\t' ); // 2-column format
76               } // end if
77            } // end for
78         } // end for
79      } // end for
80   } // end function deal
```

2. （指向函式之指標的陣列）重新撰寫圖 6.22 的程式，改為使用選單式的介面。程式應提供使用者如下的四種選項：

```
Enter a choice:
  0  Print the array of grades
  1  Find the minimum grade
  2  Find the maximum grade
  3  Print the average on all tests for each student
  4  End program
```

使用指向函式之指標的陣列有一項限制，那便是所有的指標必須具有相同的型別。也就是說，這些指標所指到的函式，其回傳型別、及接收的引數型別都必須相同。因此，圖 6.22 裡的函式都必須修改成回傳型別相同，參數型別也相同。請將函式 `minimum` 和 `maximum` 修改成印出最小和最大的數值，且不傳回任何值。對於選項 3，修改圖 6.22 的 `average` 函式，使之印出每個學生的平均成績（而非針對某位學生）。函式 `average` 必須沒有回傳值，而且它的參數必須和 `printArray`、`minimum` 和 `maximum` 函式一樣。請將指向這四個函式的指標存在 `processGrades` 陣列裡，然後以使用者輸入的選擇作為

陣列索引，來呼叫每一個函式。

```c
// Fig. 6.22: fig06_22.c
// Double-subscripted array manipulations.
#include <stdio.h>
#define STUDENTS 3
#define EXAMS 4

// function prototypes
int minimum( int grades[][ EXAMS ], size_t pupils, size_t tests );
int maximum( int grades[][ EXAMS ], size_t pupils, size_t tests );
double average( const int setOfGrades[], size_t tests );
void printArray( int grades[][ EXAMS ], size_t pupils, size_t tests );

// function main begins program execution
int main( void )
{
   size_t student; // student counter

   // initialize student grades for three students (rows)
   int studentGrades[ STUDENTS ][ EXAMS ] =
      { { 77, 68, 86, 73 },
        { 96, 87, 89, 78 },
        { 70, 90, 86, 81 } };

   // output array studentGrades
   puts( "The array is:" );
   printArray( studentGrades, STUDENTS, EXAMS );

   // determine smallest and largest grade values
   printf( "\n\nLowest grade: %d\nHighest grade: %d\n",
      minimum( studentGrades, STUDENTS, EXAMS ),
      maximum( studentGrades, STUDENTS, EXAMS ) );

   // calculate average grade for each student
   for ( student = 0; student < STUDENTS; ++student ) {
      printf( "The average grade for student %u is %.2f\n",
         student, average( studentGrades[ student ], EXAMS ) );
   } // end for
} // end main

```

```
40  // Find the minimum grade
41  int minimum( int grades[][ EXAMS ], size_t pupils, size_t tests )
42  {
43     size_t i; // student counter
44     size_t j; // exam counter
45     int lowGrade = 100; // initialize to highest possible grade
46
47     // loop through rows of grades
48     for ( i = 0; i < pupils; ++i ) {
49
50        // loop through columns of grades
51        for ( j = 0; j < tests; ++j ) {
52
53           if ( grades[ i ][ j ] < lowGrade ) {
54              lowGrade = grades[ i ][ j ];
55           } // end if
56        } // end inner for
57     } // end outer for
58
59     return lowGrade; // return minimum grade
60  } // end function minimum
61
62  // Find the maximum grade
63  int maximum( int grades[][ EXAMS ], size_t pupils, size_t tests )
64  {
65     size_t i; // student counter
66     size_t j; // exam counter
67     int highGrade = 0; // initialize to lowest possible grade
68
69     // loop through rows of grades
70     for ( i = 0; i < pupils; ++i ) {
71
72        // loop through columns of grades
73        for ( j = 0; j < tests; ++j ) {
74
75           if ( grades[ i ][ j ] > highGrade ) {
76              highGrade = grades[ i ][ j ];
77           } // end if
78        } // end inner for
79     } // end outer for
80
81     return highGrade; // return maximum grade
82  } // end function maximum
83
```

```
84  // Determine the average grade for a particular student
85  double average( const int setOfGrades[], size_t tests )
86  {
87     size_t i; // exam counter
88     int total = 0; // sum of test grades
89
90     // total all grades for one student
91     for ( i = 0; i < tests; ++i ) {
92        total += setOfGrades[ i ];
93     } // end for
94
95     return ( double ) total / tests; // average
96  } // end function average
97
98  // Print the array
99  void printArray( int grades[][ EXAMS ], size_t pupils, size_t tests )
100 {
101    size_t i; // student counter
102    size_t j; // exam counter
103
104    // output column heads
105    printf( "%s", "                [0]  [1]  [2]  [3]" );
106
107    // output grades in tabular format
108    for ( i = 0; i < pupils; ++i ) {
109
110       // output label for row
111       printf( "\nstudentGrades[%d] ", i );
112
113       // output grades for one student
114       for ( j = 0; j < tests; ++j ) {
115          printf( "%-5d", grades[ i ][ j ] );
116       } // end inner for
117    } // end outer for
118 } // end function printArray
```

```
The array is:
                [0]  [1]  [2]  [3]
studentGrades[0] 77   68   86   73
studentGrades[1] 96   87   89   78
studentGrades[2] 70   90   86   81

Lowest grade: 68
Highest grade: 96
The average grade for student 0 is 76.00
The average grade for student 1 is 87.50
The average grade for student 2 is 81.75
```

1. ***(Card Shuffling and Dealing)*** Modify the program in Fig. 7.24 so that the card-dealing function deals a five-card poker hand. Then write the following additional functions:
   a) Determine whether the hand contains a pair.
   b) Determine whether the hand contains two pairs.
   c) Determine whether the hand contains three of a kind (e.g., three jacks).
   d) Determine whether the hand contains four of a kind (e.g., four aces).
   e) Determine whether the hand contains a flush (i.e., all five cards of the same suit).
   f) Determine whether the hand contains a straight (i.e., five cards of consecutive face values).

```c
// Fig. 7.24: fig07_24.c
// Card shuffling and dealing.
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define SUITS 4
#define FACES 13
#define CARDS 52

// prototypes
void shuffle( unsigned int wDeck[][ FACES ] ); // shuffling modifies wDeck
void deal( unsigned int wDeck[][ FACES ], const char *wFace[],
   const char *wSuit[] ); // dealing doesn't modify the arrays

int main( void )
{
```

```
18      // initialize suit array
19      const char *suit[ SUITS ] =
20         { "Hearts", "Diamonds", "Clubs", "Spades" };
21
22      // initialize face array
23      const char *face[ FACES ] =
24         { "Ace", "Deuce", "Three", "Four",
25           "Five", "Six", "Seven", "Eight",
26           "Nine", "Ten", "Jack", "Queen", "King" };
27
28      // initialize deck array
29      unsigned int deck[ SUITS ][ FACES ] = { 0 };
30
31      srand( time( NULL ) ); // seed random-number generator
32
33      shuffle( deck ); // shuffle the deck
34      deal( deck, face, suit ); // deal the deck
35   } // end main
36
37   // shuffle cards in deck
38   void shuffle( unsigned int wDeck[][ FACES ] )
39   {
40      size_t row; // row number
41      size_t column; // column number
42      size_t card; // counter
43
44     // for each of the cards, choose slot of deck randomly
45     for ( card = 1; card <= CARDS; ++card ) {
46
47        // choose new random location until unoccupied slot found
48        do {
49           row = rand() % SUITS;
50           column = rand() % FACES;
51        } while( wDeck[ row ][ column ] != 0 ); // end do...while
52
53         // place card number in chosen slot of deck
54         wDeck[ row ][ column ] = card;
55      } // end for
56   } // end function shuffle
57
58   // deal cards in deck
59   void deal( unsigned int wDeck[][ FACES ], const char *wFace[],
60      const char *wSuit[] )
61   {
62      size_t card; // card counter
63      size_t row; // row counter
64      size_t column; // column counter
65
```

```
66      // deal each of the cards
67      for ( card = 1; card <= CARDS; ++card ) {
68         // loop through rows of wDeck
69         for ( row = 0; row < SUITS; ++row ) {
70            // loop through columns of wDeck for current row
71            for ( column = 0; column < FACES; ++column ) {
72               // if slot contains current card, display card
73               if ( wDeck[ row ][ column ] == card ) {
74                  printf( "%5s of %-8s%c", wFace[ column ], wSuit[ row ],
75                     card % 2 == 0 ? '\n' : '\t' ); // 2-column format
76               } // end if
77            } // end for
78         } // end for
79      } // end for
80   } // end function deal
```

2. ***(Arrays of Pointers to Functions)*** Rewrite the program of Fig. 6.22 to use a menu-driven interface. The program should offer the user four options as follows:

```
Enter a choice:
  0  Print the array of grades
  1  Find the minimum grade
  2  Find the maximum grade
  3  Print the average on all tests for each student
  4  End program
```

One restriction on using arrays of pointers to functions is that all the pointers must have the same type. The pointers must be to functions of the same return type that receive arguments of the same type. For this reason, the functions in Fig. 6.22 must be modified so that they each return the same type and take the same parameters. Modify functions minimum and maximum to print the minimum or maximum value and return nothing. For option 3, modify function average of Fig. 6.22 to output the average for each student (not a specific student). Function average should return nothing and take the same parameters as printArray, minimum and maximum. Store the pointers to the four functions in array processGrades and use the choice made by the user as the index into the array for calling each function.

```
1  // Fig. 6.22: fig06_22.c
2  // Double-subscripted array manipulations.
3  #include <stdio.h>
4  #define STUDENTS 3
5  #define EXAMS 4
```

```
40 // Find the minimum grade
41 int minimum( int grades[][ EXAMS ], size_t pupils, size_t tests )
42 {
43    size_t i; // student counter
44    size_t j; // exam counter
45    int lowGrade = 100; // initialize to highest possible grade
46
47    // loop through rows of grades
48    for ( i = 0; i < pupils; ++i ) {
49
50       // loop through columns of grades
51       for ( j = 0; j < tests; ++j ) {
52
53          if ( grades[ i ][ j ] < lowGrade ) {
54             lowGrade = grades[ i ][ j ];
55          } // end if
56       } // end inner for
57    } // end outer for
58
59    return lowGrade; // return minimum grade
60 } // end function minimum
61
62 // Find the maximum grade
63 int maximum( int grades[][ EXAMS ], size_t pupils, size_t tests )
64 {
65    size_t i; // student counter
66    size_t j; // exam counter
67    int highGrade = 0; // initialize to lowest possible grade
68
69    // loop through rows of grades
70    for ( i = 0; i < pupils; ++i ) {
71
72       // loop through columns of grades
73       for ( j = 0; j < tests; ++j ) {
74
75          if ( grades[ i ][ j ] > highGrade ) {
76             highGrade = grades[ i ][ j ];
77          } // end if
78       } // end inner for
79    } // end outer for
80
81    return highGrade; // return maximum grade
82 } // end function maximum
83
```

```
84  // Determine the average grade for a particular student
85  double average( const int setOfGrades[], size_t tests )
86  {
87     size_t i; // exam counter
88     int total = 0; // sum of test grades
89
90     // total all grades for one student
91     for ( i = 0; i < tests; ++i ) {
92        total += setOfGrades[ i ];
93     } // end for
94
95     return ( double ) total / tests; // average
96  } // end function average
97
98  // Print the array
99  void printArray( int grades[][ EXAMS ], size_t pupils, size_t tests )
100 {
101    size_t i; // student counter
102    size_t j; // exam counter
103
104    // output column heads
105    printf( "%s", "                [0]  [1]  [2]  [3]" );
106
107    // output grades in tabular format
108    for ( i = 0; i < pupils; ++i ) {
109
110       // output label for row
111       printf( "\nstudentGrades[%d] ", i );
112
113       // output grades for one student
114       for ( j = 0; j < tests; ++j ) {
115          printf( "%-5d", grades[ i ][ j ] );
116       } // end inner for
117    } // end outer for
118 } // end function printArray
```

```
The array is:
                [0]  [1]  [2]  [3]
studentGrades[0] 77   68   86   73
studentGrades[1] 96   87   89   78
studentGrades[2] 70   90   86   81

Lowest grade: 68
Highest grade: 96
The average grade for student 0 is 76.00
The average grade for student 1 is 87.50
The average grade for student 2 is 81.75
```