

## lab07

```
$ gcc -DN=11 lab07.c
```

```
$ ./a.out < mat11.in
```

```
Matrix A is_
```

```
Matrix A is
```

```
11 10 9 8 7 6 5 4 3 2 1
10 11 10 9 8 7 6 5 4 3 2
9 10 11 10 9 8 7 6 5 4 3
8 9 10 11 10 9 8 7 6 5 4
7 8 9 10 11 10 9 8 7 6 5
6 7 8 9 10 11 10 9 8 7 6
5 6 7 8 9 10 11 10 9 8 7
4 5 6 7 8 9 10 11 10 9 8
3 4 5 6 7 8 9 10 11 10 9
2 3 4 5 6 7 8 9 10 11 10
1 2 3 4 5 6 7 8 9 10 11
```

```
det(A) = 6144
```

```
CPU time: 1.27673 sec
```

```
score: 83
```

```
o. [Output] Program output is incorrect
```

```
o. [Format] Program format can be improved
```

```
o. [Coding] lab07.c spelling errors: amd(1), lexicographic(2), matirx(1), swapings(2)
```

## lab07.c

```
1 // EE231000 Lab07 Matrix Determinant
2 // 109061158, 簡佳吟
3 // Date: 2020/11/16
4 // Need a blank line here.
5 #include <stdio.h>
6 #if !defined(N)
7 #define N 3
8 #endif
9
10 int Pandita(int P[N]);
11 // This function generate the next lexicographic permutation
12 // This function generate the next lexicographic permutation
13 // based on Pandita algorithm
14 // based on Pandita algorithm
15 // input : P contains the previous permutation
16 // input : P contains the previous permutation
17 // output : return sgn of cumulative number of swapings
18 // output : return sgn of cumulative number of swapings
19 // and return 0 if no more permutation
20 // and return 0 if no more permutation
21 // P contains the next permutation
22 // P contains the next permutation
23 // Need a blank line here.
24
25 int main(void) {
26     int main(void)
27     {
28         int A[N][N];          // two dimension array to store the given matrix
29         int P[N];             // one dimension array for Pandita
30         int i, j, k;          // index for loop
31         long int sum = 0;      // sum the whole product
32         long int product = 1;  // product of each array
33         long int product = 1; // product of each array
34         int sgn = 1;           // show either positive or negative
35         int flag = 1;          // a switch to continue or stop loop
36         int flag = 1;         // a switch to continue or stop loop
37
38         printf("Matrix A is \n");          // prompt
39         for (j = 0; j < N; j++) {
40             printf(" ");
41             for (k = 0; k < N; k++) {
```

```

29         scanf(" %d", &A[j][k]);
30         printf(" %d", A[j][k]); // scan the given matrix
31     }
32     printf("\n");
33
34 }
35
36 for (i = 0; i < N; i++) {          // initialize the array
37     P[i] = i + 1;
38 }
39
40 while (flag) {
41     product = 1;                    // initialize the product
42     for (i = 0; i < N; i++) {
43         product *= A[i][P[i] - 1]; // product each array
44     }
45     if (sgn == 0) {                 // stop the loop
46         flag = 0;                  // when no more permutation
47         flag = 0;                  // when no more permutation
48     }
49     if (flag) {
50         sum += sgn * product;       // sum the product
51     }
52     sgn = Pandita(P);               // call the function Pandita
53 }
54 printf("det(A) = %ld\n", sum);     // prompt
55
56 return 0;                          // done and return
57 }
58
59
60
61 int Pandita(int P[N]) {
62     int Pandita(int P[N])
63     {
64         // This function generate the next lexicographic permutation
65         // This function generate the next lexicographic permutation
66         // based on Pandita algorithm
67         // based on Pandita algorithm
68         // input : P contains the previous permutation

```

```

        // input : P contains the previous permutation
65 // output : return sgn of cumulative number of swapings
        // output : return sgn of cumulative number of swapings
66 // and return 0 if no more permutation
        // and return 0 if no more permutation
67 // P contains the next permutation
        // P contains the next permutation
68     int i, j, k;                // index for loop
69     int max;                    // index for loop
70     int temp;                   // store number temporarily
71     int static count = 0;
    Need a blank line here.
72     for (i = N - 2; P[i] > P[i + 1] && i >= 0; i--) ;
73         // find the largest index i
74         // s.t. P[i] < P[i + 1]
75     if ( i == -1) {              // if no more permutation
        if (i == -1) {            // if no more permutation
76         return 0;               // done amd return 0
77     }
78     for (max = N - 1; P[max] < P[i]; max--) ;
79         // find the largest index max
80         // s.t P[i] < P[max]
81     temp = P[i];                 // swap P[i] and P[max]
82     P[i] = P[max];
83     P[max] = temp;
84     count++;                     // count the number of swappings
85     for (k = i + 1, j = N - 1; k < j; k++, j--) {
86         temp = P[k];             // reverse from P[i + 1] to P[N - 1]
87         P[k] = P[j];
88         P[j] = temp;
89         count++;                 // cumulate the number of swappings
90     }
91     if (count % 2 == 0) {        // if count is an even number
92         return 1;               // done and return 1
93     }
94     else {
95         return -1;              // if count is an odd number
96     }                           // done and return -1
97
98 }
99

```

100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139

Trailing blank lines should be removed.

140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180

181  
182  
183  
184  
185