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), lexicograhic(2), matirx(1), swapings(2)
```

lab07.c

```
// EE231000 Lab07 Matrix Determinant
// 109061158, 簡佳吟
// Date: 2020/11/16
Need a blank line here.
#include <stdio.h>
#if !defined(N)
#define N 3
#endif

int Pandita(int P[N]);
// This function generate the next lexicograhic permutation
    // This function generate the next lexicograhic permutation
// based on Pandita algorithm
    // based on Pandita algorithm
// input : P contains the previous permutation
    // input : P contains the previous permutation
// output : return sgn of cumulative number of swapings
    // output : return sgn of cumulative number of swapings
// and return 0 if no more permutation
    // and return 0 if no more permutation
// P contains the next permutation
    // P contains the next permutation
Need a blank line here.
int main(void) {
int main(void)
{
    int A[N][N];        // two dimension array to store the given matirx
    int P[N];           // one dimension array for Pandita
    int i, j, k;        // index for loop
    long int sum = 0;   // sum the whole product
    long int product = 1;// product of each array
    long int product = 1; // product of each array
    int sgn = 1;        // show either positive or negative
    int flag =1;        // a switch to continue or stop loop
    int flag = 1;        // a switch to continue or stop loop

    printf("Matrix A is \n");           // prompt
    for (j = 0; j < N; j++) {
        printf("  ");
        for (k = 0; k < N; k++) {
            scanf(" %d", &A[j][k]);
            printf(" %d", A[j][k]); // scan the given matrix
        }
        printf("\n");

    }

    for (i = 0; i < N; i++) {       // initialize the array
        P[i] = i + 1;
    }

    while (flag) {
        product = 1;                    // initialize the product
        for (i = 0; i < N; i++) {
            product *= A[i][P[i] - 1];  // product each array
        }
        if (sgn == 0) {                 // stop the loop
        flag = 0;                       // when no more permutation
            flag = 0;                       // when no more permutation
        }
        if (flag) {
            sum += sgn * product;       // sum the product
        }
        sgn = Pandita(P);               // call the function Pandita

    }
    printf("det(A) = %ld\n", sum);      // prompt

    return 0;                           // done and return
}



int Pandita(int P[N]) {
int Pandita(int P[N])
{
// This function generate the next lexicograhic permutation
    // This function generate the next lexicograhic permutation
// based on Pandita algorithm
    // based on Pandita algorithm
// input : P contains the previous permutation
```

```
    // input : P contains the previous permutation
// output : return sgn of cumulative number of swapings
    // output : return sgn of cumulative number of swapings
// and return 0 if no more permutation
    // and return 0 if no more permutation
// P contains the next permutation
    // P contains the next permutation
    int i, j, k;                    // index for loop
    int max;                        // index for loop
    int temp;                       // store number temporarily
    int static count = 0;
Need a blank line here.
    for (i = N - 2; P[i] > P[i + 1] && i >= 0; i--) ;
                                    // find the largest index i
                                    // s.t. P[i] < P[i + 1]
    if ( i == -1) {                 // if no more permutation
    if (i == -1) {                 // if no more permutation
        return 0;                   // done amd return 0
    }
    for (max = N - 1; P[max] < P[i]; max--) ;
                                    // find the largest index max
                                    // s.t P[i] < P[max]
    temp = P[i];                    // swap P[i] and P[max]
    P[i] = P[max];
    P[max] = temp;
    count++;                        // count the number of swappings
    for (k = i + 1, j = N - 1; k < j; k++, j--) {
        temp = P[k];                // reverse from P[i + 1] to P[N - 1]
        P[k] = P[j];
        P[j] = temp;
        count++;                    // cumulate the number of swappings
    }
    if (count % 2 == 0) {           // if count is an even number
        return 1;                   // done and return 1
    }
    else {
        return -1;                  // if count is an odd number
    }                               // done and return -1

}
```

```
Trailing blank lines should be removed.
```