執行環境：Windows
程式語言：C
Programming homework1：

```
void fastTranspose(term *a,term *b)
{
        int i,j,startingPos[a[0].column];    /* use startingPos[] to record the original
                                              rowTerms & startingPos, array size depends
                                              on columns of matrix a*/
          b[0].row = a[0].column;
          b[0].column = a[0].row;
          b[0].value = a[0].value;
          printf("%d",a[0].value);
          if(a[0].value > 0)
          {
                        for(i = 0 ; i < a[0].column ; i++) startingPos[i] = 0;
                        for(i = 1 ; i <= a[0].value ; i++) startingPos[a[i].column]++;
                        for(i = 1 ; i < a[0].column ; i++) startingPos[i] = startingPos[i] +
startingPos[i-1];
                                                          /* the first two for-loops do
                                                          the same things to rowTerms,
                                                          then adding (i-1)-th term of
                                                          startingPos to i-th.
                                                          Now, we have the last position
                                                          of each row*/

                        for(i = a[0].value ; i >= 1 ; i--)        /*Finally, counting down to
                        {                                         the first term of a, and
                                j = startingPos[a[i].column]--;   startingPos of each row
                                b[j].row = a[i].column;           decreases with the terms of
                                b[j].column = a[i].row;           row filling in b*/
                                b[j].value = a[i].value;
                        }
          }
}
```

Programming homework2：

In the main function,

```
    int starti,startj;    //current position of knight is represented by pair(starti,startj)
    int npos,nexti[8],nextj[8],min,count = 2;       /*npos = number of next possible
                                                     square
                                                     nexti, nextj = list of next possible
                                                     square
                                                     min = index of nexti, nextj with the
                                                     minimum exits in list
                                                    count = if knight moves successfully,
                                                     then count++ */

    scanf("%d %d",&starti,&startj);             /*input for starting position*/
    chessboard[starti][startj] = 1;             /*set starting position as 1*/
                                                     /*while-loop to do the following
                                                     things: form a set of possible next
                                                     squares with Listnpos, test special
                                                     case ( npos = 0,1 or >= 7 ), find next
                                                     square with minimum number of
                                                     exits with Findnextsquare.*/
    while((npos = Listnpos(starti,startj,nexti,nextj)))
    {
            if(npos == 1)      /*possible next square is 1*/
            {
                    min = 0;
                    starti = nexti[min];
                    startj = nextj[min];
                    chessboard[starti][startj] = count++;
            }
            Else               /*possible next square is more then 1*/
            {
                min = Findnextsquare(nexti,nextj,npos);
                starti = nexti[min];
                startj = nextj[min];
                chessboard[starti][startj] = count++;
            }
    }
    if(count < 64) printf("Failed!\nUsing step: %d\nFinal position:
```

```
(%d,%d)\n",count,starti,startj);
        else printf("Succeeded!\nFinal position: (%d,%d)\n",starti,startj);
        chessboardPrint(); // output result

int Listnpos(int starti,int startj,int *nexti,int *nextj)    /* possible next squares are
{                                                              stored in nexti, nextj*/
        int i,npos = 0;

        for(i = 0; i < 8 ; i++)
        {
                if( starti+ktmovev[i] >= 0 && starti+ktmovev[i] <= 7 && startj+ktmoveh[i]
>= 0 && startj+ktmoveh[i] <= 7)                    //checking next position lies in board
                    if( !chessboard[starti+ktmovev[i]][startj+ktmoveh[i]])
                                            /*checking this position hasn't been occupied */
                        {
                                nexti[npos] = starti + ktmovev[i];
                                nextj[npos] = startj + ktmoveh[i];
                                npos++;
                        }
        }

        return npos;
}

int Findnextsquare(int *nexti,int *nextj,int npos)
{
        int i,j,exit[npos],min; // exit[] records the exits in the list of next possible squares

        for(i = 0 ; i < npos ; i++)
        {
                exit[i] = 0;
                for(j = 0; j < 8 ; j++)
                    //this part is the same to Listnpos, but just counting number of exit
                {
                        if( nexti[i]+ktmovev[j] >= 0 && nexti[i]+ktmovev[j] <= 7 &&
nextj[i]+ktmoveh[j] >= 0 && nextj[i]+ktmoveh[j] <= 7)
                            if( !chessboard[nexti[i]+ktmovev[j]][nextj[i]+ktmoveh[j]])
                                exit[i]++;
```

```
            }
        }
        min = npos-1;
        for(i = npos-2 ; i >= 0 ; i--)    if(exit[i] <= exit[min]) min = i; /*This for-loop is to
                                                                              finding the location of
                                                                              the minimum value of
                                                                              exits.
                                                                              Textbook requires us
                                                                              to take the first
                                                                              occurrence in list, so
                                                                              for-loop finds the min
                                                                              backward.*/
        return min;
}
```