```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#define NMAX 100
#define tiny 1.0E-20


void ludcmp(double A[][NMAX+1], int N, int *INDX, int D);
void lubksb(double A[][NMAX+1], int N, int *INDX,
double *B);

main()
{
      int i,j;
      int N,D,dummy=NMAX;
      double A[NMAX+1][NMAX+1];
      double B[NMAX+1];
      int INDX[NMAX+1];


      do
      {
         printf("Input N (<=%d) - dimension of the
array : ",dummy);
         scanf("%d",&N);
      } while(N>NMAX);

      for(i=1;i<=N;i++)
```

```
      {
         for(j=1;j<=N;j++)
         {
            printf("Element (%d, %d) of the LHS :
",i,j);
            scanf("%lf",&A[i][j]);
         }
      printf("Element %d of the RHS : ",i);
      scanf("%lf",&B[i]);
      }

      ludcmp(A,N,INDX,D);
      lubksb(A,N,INDX,B);

      for(i=1;i<=N;i++)
      {
         fprintf(stderr,"Element %d of the result :
%lf\n",i,B[i]);
      }

      return 0;

}


void ludcmp(double A[][NMAX+1],int N,int *INDX,int D)
{
      int i,j,k,imax;
      double VV[NMAX+1];
```

```
      double aamax,sum,dummy;

      D=1;
#pragma omp parallel for private (i,j,aamax)
      for(i=1;i<=N;i++)
      {
         aamax=0;
         for(j=1;j<=N;j++)
         {
            if (abs(A[i][j])>aamax)
               aamax=abs(A[i][j]);
         }
         if (aamax==0.0)
      #pragma omp critical
         {
            printf("Singular Matrix. \n");
            exit(1);
         }
         VV[i]=1.0/aamax;
      }

      for(j=1;j<=N;j++)
      {
         for(i=1;i<=j-1;i++)
         {
            sum=A[i][j];
         #pragma omp for private(k) reduction (-:sum)
            for(k=1;k<=i-1;k++)
            {
```

```
            sum -= A[i][k]*A[k][j];
         }
         A[i][j]=sum;
      }
      aamax=0;
#pragma omp for private (i,k,sum,dummy)
      for(i=j;i<=N;i++)
      {
         sum=A[i][j];
         for(k=1;k<=j-1;k++)
         {
            sum -= A[i][k]*A[k][j];
         }
         A[i][j]=sum;
         dummy=VV[i]*abs(sum);
         if (dummy>=aamax)
#pragma omp critical
         {
            imax=i;
            aamax=dummy;
         }
      }
      if (j!=imax)
      {
#pragma omp for private(k,dummy)
         for(k=1;k<=N;k++)
         {
            dummy=A[imax][k];
            A[imax][k]=A[j][k];
```

```
            A[j][k]=dummy;
         }
         D=-D;
         VV[imax]=VV[j];
      }
      INDX[j]=imax;
      if (A[j][j]==0.0)
         A[j][j]=tiny;
      if (j!=N)
      {
         dummy=1.0/A[j][j];
      #pragma omp parallel for private(i)
         for(i=j+1;i<=N;i++)
         {
            A[i][j]=A[i][j]*dummy;
         }
      }
   }
}


void lubksb(double A[][NMAX+1],int N,int *INDX,double *B)
{
   int ii,i,j,ll;
   double sum;

   ii=0;
   for(i=1;i<=N;i++)
     printf("THE INDEX: %d\n",INDX[i]);
```

```
for(i=1;i<=N;i++)
{
   ll=INDX[i];
   sum=B[ll];
   B[ll]=B[i];
   if (ii!=0)
   {
#pragma omp for private (j) reduction (-:sum)
      for(j=ii;j<=i-1;j++)
      {
         sum -= A[i][j]*B[j];
      }
   }
   else if (sum!=0.0)
      ii=i;

   B[i]=sum;
}

for(i=N;i>=1;i--)
{
   sum=B[i];
   if (i<N)
   {
#pragma omp for private (j) reduction (-:sum)
      for(j=i+1;j<=N;j++)
      {
         sum -= A[i][j]*B[j];
      }
```

```
            }
            B[i]=sum/A[i][i];
        }
}
```