Source code - simplex.c

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

int next_iteration_index(double *ptr, int m, int n)
{
   int i,index=0;
   for(i=1;i<n-1;i++)
   {
      if(*(ptr+(m-1)*n+i) < *(ptr+(m-1)*n+index))
      {
         index = i;
      }
   }
   if(*(ptr+(m-1)*n+index) > 0)
   {
      return -1;
   }
   else
   {
      if(*(ptr+(m-1)*n+index) == 0)
      {
         return -2;
      }
   }
   return index;
}

int pivot_index(double *ptr, int m, int n)
{
   int i,j=next_iteration_index(ptr,m,n),index=j;
   double vi = *(ptr+j),vn = *(ptr+n-1),min = vn/vi;
   for(i=1;i<m-1;i++)
   {
      vi = *(ptr+i*n+j);
      vn = *(ptr+i*n+n-1);
      if(vi>0 && (vn/vi)<min)
      {
         min = vn/vi;
         index = i*n+j;
      }
   }
   return index;
}

double * convert_pivot(double *ptr, int m, int n, int p_index)
{
   double p = *(ptr+p_index);
   p = 1/p;
   *(ptr+p_index) = p;
   return ptr;
```

```
}

double * convert_row(double *ptr, int m, int n, int p_index)
{
   double p = *(ptr+p_index),r;
   int p_row = p_index/n,p_col=p_index%n,i;
   for(i=0;i<n;i++)
   {
      if(i!=p_col)
      {
         r = *(ptr+p_row*n+i);
         r = r*p;
         *(ptr+p_row*n+i) = r;
      }
   }
   return ptr;
}

double * convert_column(double *ptr, int m, int n, int p_index)
{
   double p = *(ptr+p_index),c;
   int p_row = p_index/n,p_col = p_index%n,i;
   for(i=0;i<m;i++)
   {
      if(i!=p_row)
      {
         c = *(ptr+i*n+p_col);
         c = (-1)*c*p;
         *(ptr+i*n+p_col) = c;
      }
   }
   return ptr;
}

double * convert_others(double *ptr, int m, int n, int p_index)
{
   double p = *(ptr+p_index),c,r,s;
   int p_row = p_index/n,p_col = p_index%n,i,j;
   for(i=0;i<m;i++)
   {
      for(j=0;j<n;j++)
      {
         if(i!=p_row && j!=p_col)
         {
            r = *(ptr+p_row*n+j);
            c = *(ptr+i*n+p_col);
            s = *(ptr+i*n+j);
            s += (c*r/p);
            *(ptr+i*n+j) = s;
         }
      }
   }
```

```
    return ptr;
}

void print_array(double *ptr, int m, int n)
{
    int i,j;
    for(i=0;i<m;i++)
    {
        printf("\n");
        for(j=0;j<n;j++)
        {
            printf(" %lf ",*(ptr+i*n+j));
        }
    }
}

void print_solution(double *ptr, int *ptr2, int m, int n)
{
    int i;
    for(i=0;i<n;i++)
    {
        if(*(ptr2+i) != -1)
        {
            printf(" x%d = %lf ",*(ptr2+i)+1,*(ptr+i*n+n-1));
        }
    }
    printf("\n Other variables are non-basic, i.e, 0.");
    printf("\n Optimal solution z = %lf ",*(ptr+m*n-1));
    if (next_iteration_index(ptr,m,n)==-2)
    {
        printf("\n Alternate solution exists.");
    }
    else
    {
        printf("\n Unique optimal solution - no alternate solution.");
    }
}

int * swap_variables(int *ptr2, int m, int n, int p)
{
    int p_row = p/n, p_col = p%n;
    *(ptr2+p_col) = p_row;
    return ptr2;
}

double * simplex_solve(double *ptr, int *ptr2, int m, int n)
{
    int p,i=0;
    printf("\n\n Tableu from iteration  %d \n",i++);
    print_array(ptr,m,n);
    printf("\n\n");
    while(next_iteration_index(ptr,m,n)!=-1)
    {
        printf(" Tableu from iteration  %d \n",i++);
        p = pivot_index(ptr,m,n);
        ptr2 = swap_variables(ptr2,m,n,p);
        ptr = convert_pivot(ptr,m,n,p);
        ptr = convert_row(ptr,m,n,p);
        ptr = convert_column(ptr,m,n,p);
        ptr = convert_others(ptr,m,n,p);
        print_array(ptr,m,n);
        printf("\n\n");
    }
    print_solution(ptr,ptr2,m,n);
    return ptr;
}

void main()
{
    int m,n,i,j;
    printf("\n Enter number of unknowns (n) : ");
    scanf("%d",&n);
    printf(" Enter number of equations (m) : ");
    scanf("%d",&m);
    m++;
    n++;
    double arr[m*n];
    printf("\n");
    for(i=0;i<m-1;i++)
    {
        for(j=0;j<n-1;j++)
        {
            printf(" Input for marix A's equation %d coeffiecient of x%d : ",(i+1),(j+1));
            scanf("%lf",&arr[n*i+j]);
        }
    }
    printf("\n");
    for(i=0;i<m-1;i++)
    {
        printf(" Input for matrix B's - constant for equation %d : ",(i+1));
        scanf("%lf",&arr[n*(i+1)-1]);
    }
    printf("\n");
    for(i=0;i<n-1;i++)
    {
        printf(" Input for objective function's coefficient of x%d : ",(i+1));
        scanf("%lf",&arr[(m-1)*n+i]);
    }

    for(i=0;i<n-1;i++)
    {
        arr[(m-1)*n+i] *= (-1);
    }
    arr[m*n-1] = 0.0;
```

```
     double *ptr;
    int arr2[100], *ptr2;
    for(i=0;i<n;i++)
    {
        arr2[i] = -1;
    }
    ptr = arr;
    ptr2 = arr2;
    ptr = simplex_solve(ptr,ptr2,m,n);
}
```

Output – simplex.c

```
Enter number of unknowns (n) : 4
Enter number of equations (m) : 3

Input for marix A's equation 1 coeffiecient of x1 : 1
Input for marix A's equation 1 coeffiecient of x2 : 3
Input for marix A's equation 1 coeffiecient of x3 : 0
Input for marix A's equation 1 coeffiecient of x4 : 1
Input for marix A's equation 2 coeffiecient of x1 : 2
Input for marix A's equation 2 coeffiecient of x2 : 1
Input for marix A's equation 2 coeffiecient of x3 : 0
Input for marix A's equation 2 coeffiecient of x4 : 0
Input for marix A's equation 3 coeffiecient of x1 : 0
Input for marix A's equation 3 coeffiecient of x2 : 1
Input for marix A's equation 3 coeffiecient of x3 : 4
Input for marix A's equation 3 coeffiecient of x4 : 1

Input for matrix B's - constant for equation 1 : 4
Input for matrix B's - constant for equation 2 : 3
Input for matrix B's - constant for equation 3 : 3

Input for objective function's coefficient of x1 : 2
Input for objective function's coefficient of x2 : 4
Input for objective function's coefficient of x3 : 1
Input for objective function's coefficient of x4 : 1


Tableu from iteration  0

1.000000  3.000000  0.000000  1.000000  4.000000
2.000000  1.000000  0.000000  0.000000  3.000000
0.000000  1.000000  4.000000  1.000000  3.000000
-2.000000  -4.000000  -1.000000  -1.000000  0.000000
```

```
Tableu from iteration  1

0.333333  0.333333  0.000000  0.333333  1.333333
1.666667  -0.333333  0.000000  -0.333333  1.666667
-0.333333  -0.333333  4.000000  0.666667  1.666667
-0.666667  1.333333  -1.000000  0.333333  5.333333

Tableu from iteration  2

0.333333  0.333333  -0.000000  0.333333  1.333333
1.666667  -0.333333  -0.000000  -0.333333  1.666667
-0.083333  -0.083333  0.250000  0.166667  0.416667
-0.750000  1.250000  0.250000  0.500000  5.750000

Tableu from iteration  3

-0.200000  0.400000  0.000000  0.400000  1.000000
0.600000  -0.200000  -0.000000  -0.200000  1.000000
0.050000  -0.100000  0.250000  0.150000  0.500000
0.450000  1.100000  0.250000  0.350000  6.500000

x2 = 1.000000  x1 = 1.000000  x3 = 0.500000
Other variables are non-basic, i.e, 0.
Optimal solution z = 6.500000
Unique optimal solution - no alternate solution.
```