```
//PROBLEM  FOR SEQUENTIAL SIMPLEX METHOD//
#include<iostream.h>
#include<math.h>
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#define  MACHEPS  1e-15



class funct
{
public:
double functvalue1;
void Eval(double x1,double x2)
{
/*objective function equation*/
//functvalue1=((x1-x2)+(2*x1*x1)+(2*x1*x2)+(x2*x2));
functvalue1=pow((x1*x1+x2-11),2)+pow((x1+x2*x2-7),2);
}
}



main(void)
{
funct defunct;
double  Y[4],max,min,sum1,Yc,Ye,Yr,Yo,Xc[4],Xe[4],Xr[4],Xo[4],sum2;
double  e,xk1,xk2,xk3,total,X[4][3];
int i,l,m,n,j,h,lo,found,tmax;

  printf("\n How many dimensions: "); scanf("%d",&l);
  printf("\n Accuracy: "); scanf("%lf",&e);
  printf("\n Maximum number of iterations :  "); scanf("%d",&m);
  printf("\n Alpha(Reflection Coefficient): "); scanf("%lf",&xk1);
  printf("\n Beta(contraction Coefficient): "); scanf("%lf",&xk2);
  printf("\n Gamma(Expansion Coefficient) : "); scanf("%lf",&xk3);
  printf("\n Points needed to define initial simplex is=%d",l+1);
  printf("\n Initial guess:\n\n");

  for (i=1; i<=l+1; i++)
  {
      for (j=1; j<l+1; j++)
      {
            printf("     X(%d)(%d)= ",i,j);
            scanf("%lf",&X[i][j]);
      }
  }

   n=0;
   for(tmax=1;tmax<=m;tmax++)
   {
       for(i=1;i<=l+1;i++)
       {
        defunct.Eval(X[i][1],X[i][2]);
        Y[i]=defunct.functvalue1;
       }
        if(Y[1]>Y[2])
        {
            if(Y[1]>Y[3])
            {
                if(Y[2]>Y[3])
                {
                    max=Y[1];
                    h=1;
                    min=Y[3];
                    lo=3;
                }
                else
                {
                    max=Y[1];
                    h=1;
                    min=Y[2];
                    lo=2;
                }
```

```
    }
    else
    {
        max=Y[3];
        h=3;
        min=Y[2];
        lo=2;
    }
}
else
{
    if(Y[2]>Y[3])
    {
        if(Y[1]>Y[3])
        {
            max=Y[2];
            h=2;
            min=Y[3];
            lo=3;
        }
        else
        {
            max=Y[2];
            h=2;
            min=Y[1];
            lo=1;
        }
    }
    else
    {
        max=Y[3];
        h=3;
        min=Y[1];
        lo=1;
    }
}

//To find Centroid
sum1=0.0;
sum2=0.0;
for(i=1;i<=l+1;i++)
{
    if(i!=h)
    {
        sum1=sum1+X[i][1];
        sum2=sum2+X[i][2];
    }

}

Xo[1]=sum1/2.0;
Xo[2]=sum2/2.0;
defunct.Eval(Xo[1],Xo[2]);
Yo=defunct.functvalue1;

//Find Xr by reflection Xr=(1+alpha)Xo-(alpha*max)
for(i=1;i<l+1;i++)
{
    Xr[i]=((1+xk1)*Xo[i])-(xk1*X[h][i]);
}
defunct.Eval(Xr[1],Xr[2]);
Yr=defunct.functvalue1;
found=1;
if(Yr<min)
{
//To find Xe by expansion Xe=(gamma*Xr)+(1-gamma*Xo)
    for(i=1;i<l+1;i++)
    {
        Xe[i]=(xk3*Xr[i])+((1-xk3)*Xo[i]);
    }
    defunct.Eval(Xe[1],Xe[2]);
    Ye=defunct.functvalue1;
    if(Ye<min)
    {
        X[h][1]=Xe[1];
        X[h][2]=Xe[2];
```

```
                Y[h]=Ye;
                goto e51;
            }
            else
            {
                X[h][1]=Xr[1];
                X[h][2]=Xr[2];
                Y[h]=Yr;
                goto e51;
            }

        }
        else
        {
            for(i=1;i<=l+1;i++)
            {
                if(i!=h)
                {
                    if(Yr>Y[i]) found=1;
                    else
                    {
                        found=0;
                        break;
                    }
                }
            }
            if(found==0)
            {
                X[h][1]=Xr[1];
                X[h][2]=Xr[2];
                Y[h]=Yr;
                goto e51;
            }
            else
            {
                if(Yr>max)
                {
                //Find Xc by contraction xc=beta*Xh +(1-beta)*Xo
                    for(i=1;i<l+1;i++)
                    {
                        Xc[i]=(xk2*X[h][i])+((1-xk2)*Xo[i]);
                    }
                    defunct.Eval(Xc[1],Xc[2]);
                    Yc=defunct.functvalue1;

                    if(Yc>min)
                    {
                        for (i=1; i<=l+1; i++)
                          {
                              for (j=1; j<l+1; j++)
                                {
                                    X[i][j]=(X[i][j]+X[lo][j])/2.0;
                                }
                          }
                        for(i=1;i<=l+1;i++)
                           {
                            defunct.Eval(X[i][1],X[i][2]);
                            Y[i]=defunct.functvalue1;

                           }

                        goto e51;
                    }
                    else
                    {
                        X[h][1]=Xc[1];
                        X[h][2]=Xc[2];
                        //max=Yc;
                        Y[h]=Yc;
                        goto e51;
                    }

                }
                else
                {
                X[h][1]=Xr[1];
                X[h][2]=Xr[2];
                Y[h]=Yr;
                goto e51;
                }
            }
        }

 e51:
    total=0.0;
    for(i=1;i<=l+1;i++)
    {
        total=total+(Y[i]-Yo)*(Y[i]-Yo);
    }
    total=total/(l+1);
    total=sqrt(total);

  // Check for maximum iterations and convergence
  n++;
    //if (n>=m) break;
    if (total<e) break;
}

/*INPUTS:
   l - The dimension of function to study
   e - The convergence criteria
   m - The maximum number of iterations
   xk - A starting constant
   X[i] - Initial values of variables

OUTPUTS:
   X[i] - The locally optimum set
   Eval - The value of local maximum
   n - The number of iterations performed,*/



printf("\n\n The results are:\n\n");
printf("X(1) = %1.7f\n X(2) = %1.7f\n",X[lo][1],X[lo][2]);
defunct.Eval(X[lo][1],X[lo][2]);
Y[lo]=defunct.functvalue1;
printf("\nMinima found = %2.7f\n",Y[lo]);
printf("\n The number of iterations was %d\n\n",n);
}

// End of file
```