Appendix H

Common Library and Subroutine Files

Common library file:

os.h

Common subroutine file for ZBF, ZDP, SA, and MOL:

coops.c

Common library and subroutine files for CGSA and GA-SA Program:

gasa.h, initial.c, interfac.c, memory.c, utility.c, random.c, samodule.c, stats.c, ureport.c, utriopn.c, ugeneran.gasa, ugeneran.cgsa

```
/*-------------------------------------------------------------------------*/
/* os.c - contains defined commands and included library files for LINUX
/*-------------------------------------------------------------------------*/
#define cls     "clear"
#define copy    "cp"
#define del     "rm -f"
#define ren     "mv"
#define pth     "/"
#define echo    "echo \" ["
#include <sys/time.h>
#include <unistd.h>


/*-------------------------------------------------------------------------------------
-*/
/* coops.c - contains functions reading data, build cost coefficients, calculate cost,Ploss
*/
/*          and matrix dimension, set ramp limit, read CPU time
*/
/*-------------------------------------------------------------------------------------
-*/
int ReadData(void)
{
       Word    i,j,UnitNumber;
       char    Flag[8];

  /* Read Data From Input.Dat */
  printf("\nReading Data!!");
  fscanf(Inp,"%d", &NUnit);
  for (i=0; i < NUnit; i++)
  {
      fscanf(Inp,"%d", &UnitNumber);
      fscanf(Inp,"%d", &piece[i]);
      Member = piece[i];
      if  ((Member-1) > 0)
      {
          for (j=0; j < Member-1; j++)
          {
                      fscanf(Inp,"%lf", &P_int[i][j]);
               fscanf(Inp,"%lf", &Ah[i][j]);
               fscanf(Inp,"%lf", &Bh[i][j]);
               fscanf(Inp,"%lf", &Ch[i][j]);
          }
                      fscanf(Inp,"%lf", &Ah[i][j]);
                      fscanf(Inp,"%lf", &Bh[i][j]);
                      fscanf(Inp,"%lf", &Ch[i][j]);
      }
      else
      {
                      fscanf(Inp,"%lf", &Ah[i][0]);
                      fscanf(Inp,"%lf", &Bh[i][0]);
                      fscanf(Inp,"%lf", &Ch[i][0]);
      }
      fscanf(Inp,"%lf", &Pmax[i]);
      fscanf(Inp,"%lf", &Pmin[i]);
      fscanf(Inp,"%lf", &UR[i]);
      fscanf(Inp,"%lf", &DR[i]);
      fscanf(Inp,"%lf", &Fuel_Cost[i]);
   }

   fscanf(Inp,"%d",&MaxStep);
   for(i=0; i<NUnit; i++)fscanf(Inp,"%lf",&P_ini[i]);
   for(i=0; i<MaxStep; i++)fscanf(Inp,"%lf",&Pd[i]);
   fscanf(Inp,"%s",&Flag);

   /* Based Power */
   fscanf(Inp,"%lf",&P_Base);
   /* B[0][0]  value  */
   fscanf(Inp,"%lf",&B00);
   /* B[i][0]  vector */
   for (i=0;i<NUnit;i++) fscanf(Inp,"%lf",&B0[i]);
   /* B[i][j]  matrix */
   for (i=0;i<NUnit;i++)
   for (j=0;j<NUnit;j++)fscanf(Inp,"%lf",&B[i][j]);

   /*-------- Print for check --------*/
   printf("\n%d", NUnit);
```

```
   for (i=0; i < NUnit; i++)
   {
        printf("\t%d", piece[i]);
        Member = piece[i];
        if  ((Member-1) > 0)
             {
                                for (j=0; j < Member-1; j++)
                                { printf("\t%lf", P_int[i][j]);
                                  printf("\t%lf", Ah[i][j]);
                                  printf("\t%lf", Bh[i][j]);
                                  printf("\t%lf", Ch[i][j]);
                                }
                               printf("\t%lf", Ah[i][j]);
                               printf("\t%lf", Bh[i][j]);
                               printf("\t%lf", Ch[i][j]);
             }
        else {
                        printf("\t%lf", Ah[i][0]);
                        printf("\t%lf", Bh[i][0]);
                        printf("\t%lf", Ch[i][0]);
             }
         printf("\n%lf", Pmax[i]);
         printf("\t%lf", Pmin[i]);
         printf("\t%lf", UR[i]);
         printf("\t%lf", DR[i]);
         printf("\t%lf", Fuel_Cost[i]);
   }
   printf("\nEnter to Continue");getchar();

   /* Based Power */
   printf("\n%lf",P_Base);
   /* B[0][0]  value  */
   printf("\t%lf",B00);
   /* B[i][0]  vector */
   for (i=0;i<NUnit;i++)printf("\t%lf",B0[i]);
   /* B[i][j]  matrix */
   for (i=0;i<NUnit;i++)
   for (j=0;j<NUnit;j++)printf("\t%lf",B[i][j]);
                        printf("\nEnter to Continue");getchar();
   fclose(Inp);
   return 0;
}

void BuildCoef(void)
{       /*------------- Build Coefficient ---------------*/
        Word i,j;
        for (i=0; i < NUnit; i++) {
                        for (j=0; j < piece[i]; j++) {
                                aC[i][j]= Ah[i][j]*Fuel_Cost[i];
                                bC[i][j]= Bh[i][j]*Fuel_Cost[i];
                                cC[i][j]= Ch[i][j]*Fuel_Cost[i];
}}}

void CalDim(void)
{
   /*------------ Calculate the dimension of nxn matrix----------------*/
   Word i;
        for (i=0; i < NUnit; i++)
        {
               Range = Pmax[i]-Pmin[i];
               Dimension[i] = Range/delta_1;
               if((Dimension[i]%2)!=0)Dimension[i] += 1;
               Dimension[i] /= 2;
               if(Dimension[i]<1)Dimension[i]=1;
               printf("\nDimension = %d",Dimension[i]);
}}

void SetRampLimit(double *P,double *Prmax, double *Prmin,double *SumPrmax,double *SumPrmin)
{     Word i; *SumPrmax = *SumPrmin = 0;
               for (i=0; i < NUnit; i++)
               {               Prmin[i] = P[i]-DR[i];
                               Prmax[i] = P[i]+UR[i];
                       if (Prmin[i] < Pmin[i]) Prmin[i] = Pmin[i];
                       if (Prmax[i] > Pmax[i]) Prmax[i] = Pmax[i];
                       *SumPrmax += Prmax[i]; *SumPrmin += Prmin[i];
}}
```

```
double B_Matrix( double  *P_Actual )
{  Word         i,j;
   double       P_pu[MaxUnit];
   double       Sum1,Sum2,Sum3;

   for (i=0; i < NUnit; i++) P_pu[i]=P_Actual[i]/P_Base;
   Sum1 = Sum2 = 0.0;
   for (i=0; i < NUnit; i++)
   {   Sum2 += P_pu[i]*B0[i];
         for (j=0; j < NUnit; j++) Sum1 += P_pu[i]*B[i][j]*P_pu[j];
   }
   Sum3 = P_Base*(Sum1 + Sum2 + B00);
   return(Sum3);
}

double B_Matrix1(Word ref,double Demand_Actual,double *P_Actual,double *PowPlant,double
*SumPow)
{  Word       i,j;
   double     P_pu[MaxUnit];
   double     AA, BB, CC;
   double     aa, bb, cc;
   double     Grp0, Grp1, Grp2;
   double     SumP_pu, SumP_Actual, PLmax, PLmin;
   double     Demand_pu,PLoss,root1,root2,inside_root,squareroot;

       SumP_pu   = 0.0;
       Demand_pu = Demand_Actual/P_Base;
       for (i=0; i < NUnit; i++) P_pu[i]  =  P_Actual[i]/P_Base;
       for (i=0; i < NUnit; i++) if (i!=ref) SumP_pu += P_pu[i];
       /* AA.Pr^2 + BB.Pr + CC = PL */
       Grp0 = Grp1 = Grp2 = 0.0;
       for (i=0; i<NUnit; i++)
       {   if (i!=ref) {
               Grp0 += (B[ref][i] + B[i][ref])*P_pu[i];
               Grp1 +=  B0[i]*P_pu[i];
               for (j=0; j<NUnit; j++)
                   if (j!=ref) Grp2 += B[i][j]*P_pu[i]*P_pu[j];
           }
       }
       AA = B[ref][ref];
       BB = Grp0 + B0[ref];
       CC = Grp1 + Grp2 + B00;

       P_pu[ref] = Prmin[ref]/P_Base;  PLmin  = AA*(P_pu[ref]*P_pu[ref]) + BB*P_pu[ref] +
CC;
       P_pu[ref] = Prmax[ref]/P_Base;  PLmax  = AA*(P_pu[ref]*P_pu[ref]) + BB*P_pu[ref] +
CC;
       PLmin  *= P_Base;               PLmax *= P_Base;

       /* a.Pr^2 + b.Pr + c = 0 */
       aa = AA; bb = BB-1; cc = CC + Demand_pu - SumP_pu;
       inside_root = bb*bb - 4*aa*cc;
       SumP_Actual = SumP_pu * P_Base;

       if ( inside_root < 0 ) {

            system(cls);
            printf("No real solution in quadratic equation!!\a\n");
            getchar(); getchar();
            if  (fabs(Demand_Actual+PLmax-SumP_Actual-Prmax[ref])
               < fabs(Demand_Actual+PLmin-SumP_Actual-Prmin[ref]))
                 P_Actual[ref] = Prmax[ref];
            else P_Actual[ref] = Prmin[ref];

       } else {
            squareroot = sqrt(inside_root);
            root1=(-bb-squareroot)/(2*aa);  root1 *= P_Base;
            root2=(-bb+squareroot)/(2*aa);  root2 *= P_Base;
                 if ((Prmin[ref] <= root1) && (root1 <= Prmax[ref]))
                      P_Actual[ref] = root1;
            else if ((Prmin[ref] <= root2) && (root2 <= Prmax[ref]))
                      P_Actual[ref] = root2;
            else if (fabs(Demand_Actual+PLmax-SumP_Actual-Prmax[ref])
                   < fabs(Demand_Actual+PLmin-SumP_Actual-Prmin[ref]))
                      P_Actual[ref] = Prmax[ref];
            else      P_Actual[ref] = Prmin[ref];
       } P_pu[ref] = P_Actual[ref]/P_Base;
```

```
      *PowPlant = SumP_Actual + P_Actual[ref];
       PLoss    = AA*(P_pu[ref]*P_pu[ref]) + BB*P_pu[ref] + CC;
       PLoss   *= P_Base;
      *SumPow   = (*PowPlant) - PLoss;

   return(PLoss);
}

double CostCal(double *P)
{  double cost;
   for (i=0; i < NUnit; i++)
   {   Member = piece[i];
       /* Stair Case */
       if ((Member-1) > 0)
       {   j = -1;
           do { j++;
                if( P[i] < P_int[i][j] )
                F[i] = aC[i][j] + (bC[i][j]*P[i]) + (cC[i][j]*P[i]*P[i]);
           } while (( P[i]>= P_int[i][j] ) && ( j <= Member-2 ));

           if(P[i] >= P_int[i][Member-2])
           F[i] = aC[i][Member-1] + (bC[i][Member-1]*P[i]) + (cC[i][Member-1]*P[i]*P
[i]);
       }
       /* No Stair Case */
       else  F[i] = aC[i][0] + (bC[i][0]*P[i]) + (cC[i][0]*P[i]*P[i]);
   }
   cost=0;
   for (i = 0; i < NUnit; i++) cost+=F[i];
   return cost;
}

void write_xy(int x,int y,char *string)
{    char Command[81];
     #if   (OS == MS_DOS)
           gotoxy(x,y);printf("%s",string);
     #elif ( (OS == UNIX) || (OS==LINUX) )
           sprintf(Command,"%s%i;%iH%s\"",echo,x,y,string);
           system(Command);
     #endif
}

void ReadTime(struct timeval str_time,struct timeval stp_time,struct timeval *diff_time)
{       int carry;
        diff_time->tv_usec = stp_time.tv_usec - str_time.tv_usec;
        if(diff_time->tv_usec < 0)
        {       diff_time->tv_usec += 1000000; carry = 1;
        }
        else carry = 0;
        diff_time->tv_sec = stp_time.tv_sec - str_time.tv_sec - carry;
        if(diff_time->tv_sec < 0)
        {       printf("checking time error, because of Y2K problem\n");
                getchar();
        }
        else carry = 0;
}

Start_Time()
{       gettimeofday(&start_time, &timz);    }

Stop_Time()
{       gettimeofday(&stop_time, &timz);     }

Sum_Time()
{
        SumTime.tv_sec += diff_time.tv_sec;
        SumTime.tv_usec+= diff_time.tv_usec;
        if(SumTime.tv_usec>=1000000)
        {       do{
                        SumTime.tv_usec-=1000000;
                        SumTime.tv_sec += 1;
                }while(SumTime.tv_usec>=1000000);
        }
}

void ClrScr()
{
```

```
    system(cls);
}


/*-------------------------------------------------------------------------*/
/* gasa.h - contains variable declarations                                 */
/*-------------------------------------------------------------------------*/
#define TypeFileName   ".txt"
#define min            0
#define max            1
#define LOSS           0
#define NOLOSS         1
#define ParaBit        16
#define MaxUnit        80
#define MaxMember      5
#define NSpecies       1
#define MaxNGenes      (MaxUnit-1)
#define MaxString      ParaBit*MaxNGenes
#define MaxHourDay     500
#define MOLPmin        0
#define MOLPmax        1
#define Tournament     1
#define RoulletWhl     2
#define EncodeType1    1
#define EncodeType2    2
typedef char           String[100];
typedef unsigned char   Flag;
typedef unsigned char   Byte;
typedef unsigned short  Word;
typedef Word     Genes;
typedef Genes*   Chromosome;              /* String Of Genes */
typedef double   MWs[MaxUnit];
typedef double   Lambda[MaxUnit];
typedef double   Coef[MaxUnit][MaxMember];
typedef Word     ListArray[MOLPmax+1][MaxUnit];
typedef struct
{       Chromosome   Chrom;              /* Chrom[MaxNGenes] */
        double       Fitness;
        Word         Generation;
        Word         Who;
}Best;
typedef struct
{       Chromosome   Chrom;
        double       Fitness;
        Word         Parent1,Parent2;
}Individual;
typedef Individual*  Population;      /* Population[MaxPop] */
typedef struct
{       Population   OldPop,NewPop;
        double       PCross, PMutation, RateBias, FitMax, SumFitness;
        Word         NCross, NMutation, NMicro, ipick;
        Best         BestFit;
}PrivateSpecies;


FILE    *outfp;
/* Load Demand Curve */
Word    HourDay, Hour;
double  PowerDemand[MaxHourDay];
/* ED1 Variables */
   /* Define Max/Min of Power Units */
      double     Pi[MaxUnit][max+1];
      double     Pr[MaxUnit][max+1];
   /* Define member of P_int in PG[i]*/
      double     P_int[MaxUnit][MaxMember-1];
   /* Define ramp rate constraint*/
      MWs        Ur,Dr;
   /* Define the coeficient of heat rate, and fuelcost */
      Coef       Ah,Bh,Ch;
      Coef       aC,bC,cC;
      double     Fuel_Cost[MaxUnit];
/* ED2 Variables */
      char       MyFolder[10];
      char       InpFileName[30],FileName[40],BestFileName[30];
      char       OutpFileName[30];
      char       loadtype[4],rampflag[3];
```

```
      String      Command;
      Flag        S,Method,NumFiles;
      Word        NUnit,Member,TempList;
      Word        MemberInUnit[MaxUnit];
      MWs         Sub,Pini,P;
      Lambda      IC;
      ListArray   List;
      double      F[MaxUnit];
      double      Cost_MOL[MOLPmax+1];
      double      P_MOL[MOLPmax+1][MaxUnit];
      double      SumPr[max+1],Demand[max+1];
      double      Diff,Cost,PowDemand,PowPlant;
      double      CostMax,CostMin,TotalCost,BestTCost;
      int         ListMol[MaxUnit];
/* GA with N Species */
      PrivateSpecies    *Species;          /* Species[NSpecies]; */
      Best        BBest;                  /* Best of Best Result */
      Flag        EncodeType;
      Word        Spc, PopSize, Epoch, NMigrate;
      Word        NGenes, LChrom, Generation, MaxGeneration, NumSpace;
      double      SeedRandom, RateConv;
      /* MOL-PMGA */
      Chromosome MOLChrom;
      Chromosome Full,Empty;
      Flag        TerminateMode;          /* Terminated Comp Flag */

      Word        Diff_BBestGen, Max_Diff_BBestGen;
      Word        First_mseed, Last_mseed, mseed, BestFile;

/* CPU Time */
      struct  timeval      start_time, stop_time, diff_time, SumTime;
      struct  timezone     timz;

/* B Matrix */
      double      B00;
      double      B0[MaxUnit];
      double      B[MaxUnit][MaxUnit];

/* Based Power */
      double      P_Base;

/* Loss  Power */
      Flag        LossFg;
      double      P_Loss,SumPow;

/*  SA */
       double  *Pdat,*N,*P_best_m,*upperbound,*lowerbound;


/*-------------------------------------------------------------------------*/
/* initial.c - Initialization of an initial GA individual                  */
/*-------------------------------------------------------------------------*/
void InitPop(PrivateSpecies *MonoSpecies, double SumPow, double *P_SA)
{    Word i,j,k,mask=1;
         /* keep the result from SA */
       Encode(P_SA,MonoSpecies->OldPop[0].Chrom);
       MonoSpecies->OldPop[0].Fitness = ObjFunc(SumPow,CostFunc(P_SA));

        for (i=1; i < PopSize; i++)
        {    for (j=0; j < NGenes; j++)
          {    MonoSpecies->OldPop[i].Chrom[j]=0;
               for (k=0; k < ParaBit; k++)
               {    MonoSpecies->OldPop[i].Chrom[j] <<= 1;
                    if (flip(MonoSpecies->RateBias))
                        MonoSpecies->OldPop[i].Chrom[j] |= mask;
                }
           }
           Decode(MonoSpecies->OldPop[i].Chrom,
                  P, IC, &PowPlant, &P_Loss, &SumPow);
           MonoSpecies->OldPop[i].Fitness = ObjFunc(SumPow,CostFunc(P));
     }
     Statistics(PopSize, MonoSpecies->OldPop,     &(MonoSpecies->FitMax),
                      &(MonoSpecies->SumFitness),&(MonoSpecies->BestFit));
}
```

```
/*-------------------------------------------------------------------------*/
/* interfac.c - contains subroutine functions                              */
/*-------------------------------------------------------------------------*/
#include <math.h>
Word Round(double inumber)
{
     if  ((inumber-(Word)(inumber)) >= 0.5)
           inumber=(Word)(++inumber);
    else   inumber=(Word) inumber;
    return(inumber);
}

void BuildCoef()
{
     Word i,j;
     for (i=0; i < NUnit; i++) {
          for (j=0; j < MemberInUnit[i]; j++) {
               aC[i][j]= Ah[i][j]*Fuel_Cost[i];
               bC[i][j]= Bh[i][j]*Fuel_Cost[i];
               cC[i][j]= Ch[i][j]*Fuel_Cost[i];
          }
     }
}

double B_Matrix( double  *P_Actual )
{  Word         i,j;
   MWs          P_pu;
   double       Sum1,Sum2,Sum3;

  if (LossFg == LOSS)
   {   for (i=0; i < NUnit; i++) P_pu[i]=P_Actual[i]/P_Base;
       Sum1 = Sum2 = 0.0;
       for (i=0; i < NUnit; i++)
       {   Sum2 += P_pu[i]*B0[i];
           for (j=0; j < NUnit; j++) Sum1 += P_pu[i]*B[i][j]*P_pu[j];
       }
       Sum3 = P_Base*(Sum1 + Sum2 + B00);
   }
   else  Sum3=0;
   return(Sum3);
}

double B_Matrix1( ref, Demand_Actual, P_Actual, PowPlant, SumPow)
Word    ref;
double  Demand_Actual, *P_Actual, *PowPlant, *SumPow;
{  Word       i,j;
   double     P_pu[MaxUnit];
   double     AA, BB, CC;
   double     aa, bb, cc;
   double     Grp0, Grp1, Grp2;
   double     SumP_pu, SumP_Actual, PL[max+1];
   double     Demand_pu,PLoss,root1,root2,inside_root,squareroot;

      SumP_pu   = 0.0;
       Demand_pu = Demand_Actual/P_Base;
       for (i=0; i < NUnit; i++) P_pu[i]  =  P_Actual[i]/P_Base;
       for (i=0; i < NUnit; i++) if (i!=ref) SumP_pu += P_pu[i];
       /* AA.Pr^2 + BB.Pr + CC = PL */
       Grp0 = Grp1 = Grp2 = 0.0;
       for (i=0; i<NUnit; i++)
       {   if (i!=ref) {
               Grp0 += (B[ref][i] + B[i][ref])*P_pu[i];
               Grp1 +=  B0[i]*P_pu[i];
               for (j=0; j<NUnit; j++)
                   if (j!=ref) Grp2 += B[i][j]*P_pu[i]*P_pu[j];
           }
       }
       AA = B[ref][ref];
       BB = Grp0 + B0[ref];
       CC = Grp1 + Grp2 + B00;

       P_pu[ref] = Pr[ref][min]/P_Base; PL[min]= AA*(P_pu[ref]*P_pu[ref]) + BB*P_pu[ref] +
CC;
       P_pu[ref] = Pr[ref][max]/P_Base; PL[max]= AA*(P_pu[ref]*P_pu[ref]) + BB*P_pu[ref] +
CC;
       PL[min]  *= P_Base;              PL[max] *= P_Base;
```

```
        /* a.Pr^2 + b.Pr + c = 0 */
        aa = AA; bb = BB-1; cc = CC + Demand_pu - SumP_pu;
        inside_root = bb*bb - 4*aa*cc;
        SumP_Actual = SumP_pu * P_Base;

        if ( inside_root < 0 ) {

             system(cls);
             printf("No real solution in quadratic equation!!\a\n");
             getchar(); getchar();
             if  (fabs(Demand_Actual+PL[max]-SumP_Actual-Pr[ref][max])
                < fabs(Demand_Actual+PL[min]-SumP_Actual-Pr[ref][min]))
                  P_Actual[ref] = Pr[ref][max];
             else P_Actual[ref] = Pr[ref][min];

        } else {
             squareroot = sqrt(inside_root);
             root1=(-bb-squareroot)/(2*aa);  root1 *= P_Base;
             root2=(-bb+squareroot)/(2*aa);  root2 *= P_Base;
                  if ((Pr[ref][min] <= root1) && (root1 <= Pr[ref][max]))
                       P_Actual[ref] = root1;
             else if ((Pr[ref][min] <= root2) && (root2 <= Pr[ref][max]))
                       P_Actual[ref] = root2;
             else if (fabs(Demand_Actual+PL[max]-SumP_Actual-Pr[ref][max])
                    < fabs(Demand_Actual+PL[min]-SumP_Actual-Pr[ref][min]))
                       P_Actual[ref] = Pr[ref][max];
             else      P_Actual[ref] = Pr[ref][min];
        } P_pu[ref] = P_Actual[ref]/P_Base;

       *PowPlant = SumP_Actual + P_Actual[ref];
        PLoss    = AA*(P_pu[ref]*P_pu[ref]) + BB*P_pu[ref] + CC;
        PLoss   *= P_Base;
       *SumPow   = (*PowPlant) - PLoss;
        return(PLoss);
}

double CostFunc(double  *P)
{
  double  TotalCst;
  Word  i,j,Member;
  for (i=0; i < NUnit; i++)
  {   Member = MemberInUnit[i];
      /* Stair Case */
         if ((Member-1) > 0)
         {  j = -1;
            do { j++;
                 if  ( P[i]  <  P_int[i][j] )
                       F[i]  =  aC[i][j] + (bC[i][j]*P[i]) + (cC[i][j]*P[i]*P[i]);
            } while (( P[i] >=  P_int[i][j] ) && ( j <= Member-2 ));

            if(P[i] >= P_int[i][Member-2])
               F[i]  = aC[i][Member-1] + (bC[i][Member-1]*P[i]) + (cC[i][Member-1]*P[i]*P
[i]);
         }
      /* Non Stair Case */
      else  F[i]  = aC[i][0] + (bC[i][0]*P[i]) + (cC[i][0]*P[i]*P[i]);
  }
  TotalCst=0;
  for (i = 0; i < NUnit; i++) TotalCst=TotalCst+F[i];
  return(TotalCst);
}

double ObjFunc(double SumPow,double Cost)
{
        double Pow,Cst,Ans;
        Ans=SumPow-PowDemand;     Ans=Ans*Ans;
        Pow=1 + Ans;
        Ans=Cost-CostMin;         Ans=Ans*Ans;
        Cst=1 + (Diff*Ans);
        Ans=0.5/Cst + 0.5/Pow;
        return(Ans);
}

void Decode( Chrom, P, IC, PowPlant, P_Loss, SumPow)
Chromosome   Chrom;
double       *P, *IC;
double       *PowPlant, *P_Loss, *SumPow;
```

```
{
     Word   i, j, Member;
     for (i=0; i < NGenes; i++)           /* For i=1..N-1 Generator*/
     P[i]=Pr[i][min]+(Chrom[i]*(Pr[i][max]-Pr[i][min])/NumSpace);
     *P_Loss = B_Matrix1(NUnit-1,PowDemand,P,PowPlant,SumPow);
     /* Incremental Function*/
     for (i=0; i < NUnit; i++)
     {   Member = MemberInUnit[i];
          /* Stair Case */
          if ((Member-1) > 0)
          {   j = -1;
              do { j++;
                   if ( P[i] <  P_int[i][j] )   IC[i] = bC[i][j] + (2*cC[i][j]*P[i]);
                 } while((P[i] >= P_int[i][j] ) &&  (j <=Member-2));
                if (P[i] >=  P_int[i][Member-2])
                IC[i]  =  bC[i][Member-1] + (2*cC[i][Member-1]*P[i]);
              }
             /* Non Stair Case */
             else  IC[i] = bC[i][0] + (2*cC[i][0]*P[i]);
           }
     }

void Encode(double *P,Chromosome Chrom)
{
     Word i;
     for (i=0; i < NGenes; i++)  /* For i=1..N-1 Generator */
     Chrom[i] = Round((double)NumSpace*(P[i]-Pr[i][min])/(Pr[i][max]-Pr[i][min]));
}


/*-------------------------------------------------------------------------*/
/* memory.c - reserves and releases the array location in memory           */
/*-------------------------------------------------------------------------*/
int  nomemory(string)
char *string;
{
  printf("malloc: out of memory making %s!!\n\a",string);
  fprintf(outfp,"malloc: out of memory making %s!!\n",string);
  getchar(); exit(-1);
}
void initmalloc_sga()
/* memory allocation of space for global data structures */
{
  Word       Spc,j,nbytes;
  /* memory for Species of populations */
  nbytes = NSpecies*sizeof(PrivateSpecies);
  if ( (Species = (PrivateSpecies *) malloc(nbytes)) == NULL )
                nomemory(stderr,"Species");
  /* memory for old and new populations of individuals */
  nbytes = PopSize*sizeof(Individual);
  for (Spc=0; Spc < NSpecies; Spc++)
  {     if( (Species[Spc].OldPop = (Population) malloc(nbytes)) == NULL )
                nomemory(stderr,"OldPop");
        if( (Species[Spc].NewPop = (Population) malloc(nbytes)) == NULL )
                nomemory(stderr,"NewPop");
  }
  /* memory for chromosome strings in populations */
  nbytes = NGenes*sizeof(Genes);
  for (Spc = 0; Spc < NSpecies; Spc++)
  {   for (j = 0; j < PopSize; j++)
      {   if( (Species[Spc].OldPop[j].Chrom = (Chromosome) malloc(nbytes)) == NULL )
                nomemory(stderr,"OldPop Chromosomes");
          if( (Species[Spc].NewPop[j].Chrom = (Chromosome) malloc(nbytes)) == NULL )
                nomemory(stderr,"NewPop Chromosomes");
      }
      if( (Species[Spc].BestFit.Chrom = (Chromosome) malloc(nbytes)) == NULL)
              nomemory(stderr,"BestFit Chromosome");
 }
 nbytes = NGenes*sizeof(Genes);
 if((BBest.Chrom = (Chromosome) malloc(nbytes)) == NULL)nomemory(stderr,"BBest
Chromosome");
 if((MOLChrom = (Chromosome) malloc(nbytes)) == NULL)nomemory(stderr,"MOL   Chromosome");
 if((Empty = (Chromosome) malloc(nbytes)) == NULL)nomemory(stderr,"Empty Chromosome");
 if((Full  = (Chromosome) malloc(nbytes)) == NULL)nomemory(stderr,"Full  Chromosome");

 nbytes = MaxUnit*sizeof(double);
 if((Pdat = (double *)malloc(nbytes))==NULL)nomemory(stderr,"Pdat");
```

```
 if((N = (double *)malloc(nbytes))==NULL)nomemory(stderr,"N");
 if((P_best_m = (double *)malloc(nbytes))==NULL)nomemory(stderr,"P_best_m");
 if((upperbound = (double *)malloc(nbytes))==NULL)nomemory(stderr,"upperbound");
 if((lowerbound = (double *)malloc(nbytes))==NULL)nomemory(stderr,"lowerbound");

}

void freeall_sga()
/* A routine to free all the space dynamically allocated in initspace() */
{
  Word i,Spc;
  free(Empty);
  free(Full);
  free(MOLChrom);
  free(BBest.Chrom);
  for(Spc=0;Spc< NSpecies;Spc++)
  {
        for(i = 0; i < PopSize; i++)
        {
                free(Species[Spc].OldPop[i].Chrom);
                free(Species[Spc].NewPop[i].Chrom);
        }
        free(Species[Spc].OldPop);
        free(Species[Spc].NewPop);
        free(Species[Spc].BestFit.Chrom);
  }
  free(Species);
  free(Pdat);
  free(N);
  free(P_best_m);
  free(upperbound);
  free(lowerbound);

}


/*-------------------------------------------------------------------------*/
/* random.c - contains random number generator and related utilities,      */
/* including advance_random, warmup_random, random, randomize, flip, and rnd  */
/*-------------------------------------------------------------------------*/
/* variables are declared static so that they cannot conflict with names of   */
/* other global variables in other files.  See K&R, p 80, for scope of static */
static double  oldrand[55];                     /* Array of 55 random numbers */
static Word     jrand;                                /* current random number */

void advance_random()
{
     Word   j1;
     double new_random;
     for( j1=0; j1<24;j1++)
     {
         new_random = oldrand[j1] - oldrand[j1+31];
         if(new_random < 0.0) new_random = new_random + 1.0;
         oldrand[j1] = new_random;
     }
     for(j1 = 24; j1 < 55; j1++)
     {
         new_random = oldrand [j1] - oldrand [j1-24];
         if(new_random < 0.0) new_random = new_random + 1.0;
         oldrand[j1] = new_random;
     }
}
double randomperc()
{
       jrand++;
       if(jrand >= 55)
       {  jrand = 1;
          advance_random();
       }
       return(oldrand[jrand]);
}

Word flip(double prob)
{
     double randomperc();
     if (randomperc() <= prob)
         return(1);
```

```
    else
        return(0);
}

/* Get seed number for random and start it up */
void Randomize()
{
    double randomseed;
    Word   j1;
    for(j1=0; j1<=54; j1++)  oldrand[j1] = 0.0;
    jrand=0;
    do
    {   printf(" Enter random number seed, 0.0 to 1.0 -> ");
       scanf("%lf", &randomseed);
    } while((randomseed < 0.0) || (randomseed > 1.0));
    warmup_random(randomseed);
}

Word rnd(Word low,Word high)
{
    Word    i;
    double  randomperc();
    if(low >= high)
          i = low;
    else
    {     i = (randomperc() * (high - low + 1)) + low;
          if(i > high) i = high;
    }
    return(i);
}

short rnds(short low,short high)
{
     short  i;
     double randomperc();
     if(low >= high)
           i = low;
     else
     {     i = (randomperc() * (high - low + 1)) + low;
           if(i > high) i = high;
     }
     return(i);
}

double rndreal(double lo ,double hi)
{
      return((randomperc() * (hi - lo)) + lo);
}

warmup_random(double random_seed)
{
      Word    j1, ii;
      double  new_random, prev_random;

      oldrand[54] = random_seed;
      new_random  = 0.000000001;
      prev_random = random_seed;
      for(j1 = 1 ; j1 <= 54; j1++)
      {   ii = (21*j1)%54;
          oldrand[ii] = new_random;
          new_random  = prev_random-new_random;
          if(new_random<0.0) new_random = new_random + 1.0;
          prev_random = oldrand[ii];
      }
      advance_random();
      advance_random();
      advance_random();
      jrand = 0;
}


/*--------------------------------------------------------------------------*/
/* samodule.c - sa subroutine program                                       */
/*--------------------------------------------------------------------------*/
#define INF pow(10,14);
double Initial(double SumPrmax,double Demand_Actual,double *P)
{
```

```
        double P_ini[MaxUnit];
        double PowPlant,SumPow,BestCost,Cost,SumPow_Best;
        Word   i,j,ref,varyref,ineq;
        BestCost = INF;

        /*------------- for divider --------------------*/
        for(i = 0; i < NUnit; i++)
        {       P[i] = (Pr[i][max]*Demand_Actual)/(SumPrmax);
                if(P[i]<Pr[i][min])P[i] = Pr[i][min];
                if(P[i]>Pr[i][max])P[i] = Pr[i][max];
        }
        /*------------------- Pref and PLoss Calculation  ----------------------*/
        /*for initialize there is a problem with ref P in order to meet the load demand
        /*The ref P should be vary to get the best fit  */
        varyref=0;
        /*Don't want P_next_m to change during finding the best initial P*/
        for (i = 0; i < NUnit; i++)P_ini[i] = P[i];
        do{     /*ref = NUnit-1;*/
                for(ref = 0; ref < NUnit; ref++)
                {
                        B_Matrix1(ref,Demand_Actual,P_ini,&PowPlant,&SumPow);
                        /*----- if all initial P are in the ramp---------*/
                        ineq = 0;
                        for (i=0; i < NUnit; i++)
                        if((P_ini[i]<Pr[i][min])||(P_ini[i]>Pr[i][max]))ineq++;
                        /*---------- cost calculation -----------------*/
                        if ((fabs(SumPow-Demand_Actual)<0.00001)&&(ineq==0))
                        {       varyref++;
                                Cost = CostFunc(P_ini);
                                if(Cost < BestCost)
                                {       BestCost = Cost;
                                        SumPow_Best = SumPow;
                                        for(i=0;i<NUnit;i++)P[i] = P_ini[i];
                                }
                        }
                        else
                        {
                                for (i = 0; i < NUnit; i++)
                                {       if(P_ini[i]<Pr[i][min])P_ini[i] = Pr[i][min];
                                        if(P_ini[i]>Pr[i][max])P_ini[i] = Pr[i][max];
                                }
                        }
                }/*end of vary ref loop*/
                if(varyref == 0)
                {
            if((Demand_Actual-SumPow)>0)for (i=0; i<NUnit; i++)P_ini[i] *= 1.01;
            else for (i=0; i<NUnit; i++)P_ini[i] *= 0.99;
                }
          }while(varyref == 0);
      return SumPow_Best;
}

Word rndint(Word low,Word high)
{
      Word    i;
      i = ((rand()*(high - low))/RAND_MAX) + low;
      return i;
}

double SA(double Demand_Actual,double *P_Best,Word Trial,double sigma_1,double
sigma_f,double *SumPow_Best,double *PLoss_Best,Word *ref_Best)
{       double  Pi[MaxUnit];
        double  Nupper,Nlower;
        double  PLoss,PL_best_m,SumPow_m;
        double  Cost,BestCost_m,BestCost,costdiff;
        double  Pop,Gamma,r;
        Word    ref,ref_m,i,k,m,ineq;
        double  sigma_k;

        BestCost = INF;
        Gamma = 1; r=0.95;
        /*-------- do while sigma_k > 1 iterations ------------------*/
        BestCost_m = CostFunc(P_Best);
        for(i=0; i<NUnit; i++)P_best_m[i] = P_Best[i];

        srand(mseed);
```

```
        k = 1;
        do{
                /*----------------  Calculate the control parameter--------------*/
                sigma_k = pow(r,k-1)*sigma_1;
                for(i=0; i<NUnit; i++)Pi[i] = P_best_m[i];
                /*------- for m = 1000 trials -----------------*/
                for(m = 1; m <= Trial; m++)
                {
                        for(i = 0; i < NUnit; i++)
                        {       lowerbound[i] = -Gamma*sigma_k + Pi[i];
                                upperbound[i] = Pi[i]+ Gamma*sigma_k;
                                if (lowerbound[i] < Pr[i][min])lowerbound[i] = Pr[i][min];
                                if (upperbound[i] > Pr[i][max])upperbound[i] = Pr[i][max];
                        }
                        do{
                                /*--- (a) ----*/
                                /*Random number between 0 and NUnit-1*/
                                ref = rndint(0,(NUnit-1));
                                /*---- (b) ----*/
                                for(i = 0; i < NUnit; i++)N[i] = 0;
                                ineq = 0;
                                for (i = 0; i<NUnit; i++)if (i != ref)
                                {       Nupper = upperbound[i]-Pi[i];
                                        Nlower = lowerbound[i]-Pi[i];
                                        if(Nlower>0)Nlower=0;if(Nupper<0)Nupper=0;
                                        N[i] = rndreal(Nlower,Nupper);
                                        /*---- (c) ----*/
                                        Pdat[i] = Pi[i]+N[i];
                                        if((Pdat[i]>upperbound[i])||(Pdat[i]<lowerbound[i]))
++ineq;
                                }
                        } while(ineq != 0);

                        /*---- (d) ----*/
                        /*-------------------- Pref and PLoss Calculation  ---------------*/
                        PLoss = B_Matrix1(ref,Demand_Actual,Pdat,&PowPlant,&SumPow);

                        /*---------- (f) --------------*/
                        if (fabs(SumPow-Demand_Actual)<0.00001)
                        {
                                /*---------- cost calculation -----------------*/
                                Cost = CostFunc(Pdat);
                                /*-------- (g) -------------*/
                                costdiff = Cost - BestCost_m;
                                if(costdiff < 0)
                                {  BestCost_m = Cost;
                                        for (i=0; i<NUnit; i++)
{P_best_m[i] = Pdat[i];Pi[i] = Pdat[i]; }
                                        PL_best_m = PLoss;SumPow_m = SumPow;ref_m=ref;
                                }
                                else
                                {
                                        if((costdiff/sigma_k)>100)Pop = 0;
                                        else Pop = 1/(1+exp(costdiff/sigma_k));
                                        if(Pop > (rand()/RAND_MAX))
for (i=0; i<NUnit; i++) Pi[i] = Pdat[i];
                                        }
                        }
                }

                if(BestCost_m < BestCost)
                {       BestCost = BestCost_m;
                        for (i=0; i<NUnit; i++)P_Best[i] = P_best_m[i];
                        *PLoss_Best    = PL_best_m;
                        *SumPow_Best   = SumPow_m;
                        *ref_Best = ref_m;
                }
        k++;}while(sigma_k >=sigma_f);

        return BestCost;
}

/*-------------------------------------------------------------------------*/
/* stats.c - selects and keeps the best individual and keep                */
/*-------------------------------------------------------------------------*/
void Statistics(PopSize,Pop,FitMax,SumFitness,BestIndividual)
Byte PopSize;
```

```
Population Pop;
double  *FitMax,*SumFitness;
Best    *BestIndividual;
{        Word j;
        (*SumFitness) = 0.0;
        (*FitMax) = Pop[0].Fitness;
        for ( j=0; j < PopSize; j++)
        {
           (*SumFitness)  +=  Pop[j].Fitness;
            if (Pop[j].Fitness > (*FitMax))  (*FitMax) = Pop[j].Fitness;
            if (Pop[j].Fitness > BestIndividual->Fitness)
            {
               memcpy( BestIndividual->Chrom, Pop[j].Chrom, NGenes*sizeof(Genes) );
               BestIndividual->Fitness =Pop[j].Fitness;
               BestIndividual->Generation = Generation;
}}}

void Check_BestSol()
{    Word i;
     for (i=0; i < NSpecies; i++)
         if  (Species[i].BestFit.Fitness > BBest.Fitness)
              BBest = Species[i].BestFit;
}


/*-------------------------------------------------------------------------*/
/* ugeneran.cgsa - contains subroutine function for CGSA program           */
/*-------------------------------------------------------------------------*/
void Elitism(Pop,BestIndividual)
Population   Pop;
Best*   BestIndividual;
{       Word  i, j, jelitism;
        Flag  kelite;

        kelite=0;
        for (i=0; i < PopSize; i++)
        {       jelitism=0;
                for (j=0; j < NGenes; j++)
                        if (Pop[i].Chrom[j] == BestIndividual->Chrom[j]) jelitism++;

                if (jelitism == NGenes) {
                        kelite = 1;  BestIndividual->Who=i;
                }
        }

        if (kelite == 0)

        {       BestIndividual->Who=rnd(0,PopSize-1);
                memcpy( Pop[BestIndividual->Who].Chrom, BestIndividual->Chrom,
                        NGenes*sizeof(Genes) );
                Pop[BestIndividual->Who].Fitness = BestIndividual->Fitness;
        }
}

int ReplaceMent(Chromosome Parent1,Chromosome Parent2,double Parent1_Fitness,double
Parent2_Fitness,Chromosome Child1,Chromosome Child2)
{       double FitValue[4],temd,SumPow;
        Word i,m,temw,flag,ArrangeFit[4];

        for(i=0; i<4; i++)ArrangeFit[i] = i;
        FitValue[0]=Parent1_Fitness;
        FitValue[1]=Parent2_Fitness;

        Decode(Child1, P, IC, &PowPlant, &P_Loss, &SumPow);
        FitValue[2]=ObjFunc(SumPow,CostFunc(P));
        Decode(Child2, P, IC, &PowPlant, &P_Loss, &SumPow);
        FitValue[3]=ObjFunc(SumPow,CostFunc(P));
        do{     flag = 0;
                for(i=0; i<3; i++)
                if(FitValue[i]>FitValue[i+1])
                {       temw = ArrangeFit[i+1];
                        ArrangeFit[i+1] = ArrangeFit[i];
                        ArrangeFit[i] = temw;
                        temd = FitValue[i+1];
                        FitValue[i+1] = FitValue[i];
                        FitValue[i] = temd;
                        flag++;
```

```
                }
        }while(flag!=0);

        if((ArrangeFit[2]>1)&&(ArrangeFit[3]>1))
        {       /*printf("\nChildren are better --- No Change!!");*/
        }
        else if((ArrangeFit[3]==0)&&(ArrangeFit[2]==3))memcpy(Child1,Parent1,NGenes*sizeof
(NGenes));

        else if((ArrangeFit[3]==0)&&(ArrangeFit[2]==2))memcpy(Child2,Parent1,NGenes*sizeof
(NGenes));
        else if((ArrangeFit[3]==0)&&(ArrangeFit[2]==1))
        {
        memcpy(Child1,Parent1,NGenes*sizeof(NGenes));memcpy(Child2,Parent2,NGenes*sizeof
(NGenes));

        }
        else if((ArrangeFit[3]==1)&&(ArrangeFit[2]==3))memcpy(Child1,Parent2,NGenes*sizeof
(NGenes));
        else if((ArrangeFit[3]==1)&&(ArrangeFit[2]==2))memcpy(Child2,Parent2,NGenes*sizeof
(NGenes));
        else if((ArrangeFit[3]==1)&&(ArrangeFit[2]==0))
        {
        memcpy(Child1,Parent1,NGenes*sizeof(NGenes));memcpy(Child2,Parent2,NGenes*sizeof
(NGenes));
        }

        else if((ArrangeFit[2]==0)&&(ArrangeFit[3]==2))memcpy(Child2,Parent1,NGenes*sizeof
(NGenes));
        else if((ArrangeFit[2]==0)&&(ArrangeFit[3]==3))memcpy(Child1,Parent1,NGenes*sizeof
(NGenes));

        else if((ArrangeFit[2]==1)&&(ArrangeFit[3]==2))memcpy(Child2,Parent2,NGenes*sizeof
(NGenes));
        else if((ArrangeFit[2]==1)&&(ArrangeFit[3]==3))memcpy(Child1,Parent2,NGenes*sizeof
(NGenes));
}

void CGSA_Process(SelectionType,PopSize,RateConv,MonoSpecies,Tr,S1,Sf,Generation,Epoch)
Flag      SelectionType;
Word      PopSize;
double    RateConv;
PrivateSpecies*  MonoSpecies;
Word      Tr,Generation,Epoch;
double    S1,Sf;
{       Word        j,Mate1,Mate2,ref,PopNo;
        double      CostSA,Fit;
        Population  temp;
        Word    Temp[NGenes];


        j=0;    MonoSpecies->NCross = 0;
                MonoSpecies->NMutation=0;
        do {
                if (SelectionType==Tournament)
                {       Mate1=TourSelect(PopSize,MonoSpecies->OldPop,&(MonoSpecies->ipick));
                        Mate2=TourSelect(PopSize,MonoSpecies->OldPop,&(MonoSpecies->ipick));
                }
                else
                {       Mate1=RoulSelect(PopSize,MonoSpecies->OldPop,MonoSpecies->
SumFitness);
                        Mate2=RoulSelect(PopSize,MonoSpecies->OldPop,MonoSpecies->
SumFitness);
                }
                UniCrossover(   MonoSpecies->OldPop[Mate1].Chrom,
                                MonoSpecies->OldPop[Mate2].Chrom,
                                MonoSpecies->NewPop[j].Chrom,
                                MonoSpecies->NewPop[j+1].Chrom,
                                &(MonoSpecies->PCross),
                                &(MonoSpecies->NCross));

                ReplaceMent(    MonoSpecies->OldPop[Mate1].Chrom,MonoSpecies->OldPop
[Mate2].Chrom,
                                MonoSpecies->OldPop[Mate1].Fitness,MonoSpecies->OldPop
[Mate2].Fitness,
                                MonoSpecies->NewPop[j].Chrom,MonoSpecies->NewPop[j+1].Chrom);
```

```
                Mutation( MonoSpecies->NewPop[j].Chrom,
                        &(MonoSpecies->PMutation),
                        &(MonoSpecies->NMutation));

                Mutation( MonoSpecies->NewPop[j+1].Chrom,
                        &(MonoSpecies->PMutation),
                        &(MonoSpecies->NMutation));

                 Decode(MonoSpecies->NewPop[j].Chrom,
                        P, IC, &PowPlant, &P_Loss, &SumPow);
                 MonoSpecies->NewPop[j].Fitness=ObjFunc(SumPow,CostFunc(P));

                Decode(MonoSpecies->NewPop[j+1].Chrom,
                        P, IC, &PowPlant, &P_Loss, &SumPow);
                 MonoSpecies->NewPop[j+1].Fitness=ObjFunc(SumPow,CostFunc(P));

                j += 2;

        } while ( j<(PopSize-1) );

        if((Generation%Epoch)==0)
        {       /*------- Select Best from that Gen ------*/
                Fit = MonoSpecies->NewPop[0].Fitness;PopNo = 0;
                for(j=1; j<PopSize; j++)if(MonoSpecies->NewPop[j].Fitness>Fit)
                {       Fit = MonoSpecies->NewPop[j].Fitness;PopNo = j;
                }
                 Decode(MonoSpecies->NewPop[PopNo].Chrom,P, IC, &PowPlant, &P_Loss,
&SumPow);
                if(fabs(SumPow-PowDemand)>0.00001)printf("\nGen %d InFeas GA Sol Skip
SA",Generation);
                 else{  CostSA = SA(PowDemand,P,Tr,S1,Sf,&SumPow,&P_Loss,&ref);
                        Encode(P,MonoSpecies->NewPop[PopNo].Chrom);
                        MonoSpecies->NewPop[PopNo].Fitness = ObjFunc(SumPow,CostSA);

                }
        }

        Statistics( PopSize,MonoSpecies->NewPop,&(MonoSpecies->FitMax),&(MonoSpecies->
SumFitness),&(MonoSpecies->BestFit));

                        temp = MonoSpecies->OldPop;
        MonoSpecies->OldPop = MonoSpecies->NewPop;
        MonoSpecies->NewPop = temp;

        Elitism(MonoSpecies->OldPop,&(MonoSpecies->BestFit));
}

/*-------------------------------------------------------------------------*/
/* ugeneran.gasa - contains subroutine function for GA-SA program          */
/*-------------------------------------------------------------------------*/
void Elitism(Pop,BestIndividual)

Population   Pop;
Best*   BestIndividual;
{       Word  i, j, jelitism;
        Flag  kelite;

        kelite=0;
        for (i=0; i < PopSize; i++)
        {       jelitism=0;
                for (j=0; j < NGenes; j++)
                        if (Pop[i].Chrom[j] == BestIndividual->Chrom[j]) jelitism++;

                if (jelitism == NGenes) {
                        kelite = 1;  BestIndividual->Who=i;
                }
        }

        if (kelite == 0)

        {       BestIndividual->Who=rnd(0,PopSize-1);
                memcpy( Pop[BestIndividual->Who].Chrom, BestIndividual->Chrom,
                        NGenes*sizeof(Genes) );
                Pop[BestIndividual->Who].Fitness = BestIndividual->Fitness;
        }
}
```

```
void SGA_Process(SelectionType,PopSize,RateConv,MonoSpecies)
Flag      SelectionType;
Word      PopSize;
double    RateConv;
PrivateSpecies*  MonoSpecies;
{       Word        j,Mate1,Mate2;
        Population  temp;

        j=0;    MonoSpecies->NCross = 0;
                MonoSpecies->NMutation=0;
        do {
                if (SelectionType==Tournament)
                {       Mate1=TourSelect(PopSize,MonoSpecies->OldPop,&(MonoSpecies->ipick));
                        Mate2=TourSelect(PopSize,MonoSpecies->OldPop,&(MonoSpecies->ipick));
        }
        else
                {       Mate1=RoulSelect(PopSize,MonoSpecies->OldPop,MonoSpecies->
SumFitness);
                        Mate2=RoulSelect(PopSize,MonoSpecies->OldPop,MonoSpecies->
SumFitness);
                }
        UniCrossover(   MonoSpecies->OldPop[Mate1].Chrom,
                        MonoSpecies->OldPop[Mate2].Chrom,
                        MonoSpecies->NewPop[j].Chrom,
                        MonoSpecies->NewPop[j+1].Chrom,
                        &(MonoSpecies->PCross),
                        &(MonoSpecies->NCross));
        Mutation( MonoSpecies->NewPop[j].Chrom,
                &(MonoSpecies->PMutation),
                        &(MonoSpecies->NMutation));

        Mutation( MonoSpecies->NewPop[j+1].Chrom,
                        &(MonoSpecies->PMutation),
                        &(MonoSpecies->NMutation));

       Decode(MonoSpecies->NewPop[j].Chrom,P, IC, &PowPlant, &P_Loss, &SumPow);
       MonoSpecies->NewPop[j].Fitness=ObjFunc(SumPow,CostFunc(P));

        Decode(MonoSpecies->NewPop[j+1].Chrom,P, IC, &PowPlant, &P_Loss, &SumPow);
       MonoSpecies->NewPop[j+1].Fitness=ObjFunc(SumPow,CostFunc(P));

        j += 2;

        } while ( j<(PopSize-1) );
        Statistics( PopSize, MonoSpecies->NewPop,       &(MonoSpecies->FitMax),
                        &(MonoSpecies->SumFitness), &(MonoSpecies->BestFit));

                        temp = MonoSpecies->OldPop;
        MonoSpecies->OldPop = MonoSpecies->NewPop;
        MonoSpecies->NewPop = temp;

        Elitism(MonoSpecies->OldPop,    &(MonoSpecies->BestFit));
}


/*-------------------------------------------------------------------------*/
/* ureport.c - write the result files                                      */
/*-------------------------------------------------------------------------*/
void WriteChrom(Chromosome Chrom)
{
     Word j,k,mask,tmp;
     mask=0x8000;
     for (j=NGenes; j >= 1; j--) {
          if ((j%4==0) && (j!=NGenes)) printf("\n");
          tmp=Chrom[j-1];
          for (k=ParaBit; k >= 1; k--) {
               if  ((tmp & mask) == 0x8000)
                     printf("1");
               else  printf("0");
               tmp <<= 1;
          }    printf(" ");
     }
}

void WriteChrom1(Chromosome Chrom)
{    Word j,k,mask,tmp;
```

```
     for (j=NGenes; j >= 1; j--) {
          if ((j%4==0) && (j!=NGenes)) printf("\n");
          printf("%4X", Chrom[j-1]);
          printf(" ");
     }
}

void TitleReport()
{    Word i;
     double PowerPlant;
     fprintf(outfp,"Parameters\n");
     fprintf(outfp,"--------------\n\n");
     fprintf(outfp,"Population Size (PopSize)        = %i\n", PopSize);
     fprintf(outfp,"Chromosome Length (LChrom)       = %i\n", LChrom);
     fprintf(outfp,"Maximum # of Generation (MaxGen) = %i\n", MaxGeneration);
     fprintf(outfp,"Migrate Every %i Generations\n", Epoch);
     for (Spc=0; Spc < NSpecies; Spc++) {
          fprintf(outfp,"Crossover Probabitlity (PCross)  = %.4f\n", Species[Spc].PCross);
          fprintf(outfp,"Mutation Probatility (PMutation) = %.4f\n", Species
[Spc].PMutation);
     }
     fprintf(outfp,"Seed Random (SeedRandom)         = %.4f\n", SeedRandom);
     fprintf(outfp,"RateConvergence (RateConv)       = %.4f\n", RateConv);
     skip(outfp,1);
     fprintf(outfp,"Constraint of Power Generation\n");
for (i=0; i < NUnit; i++) fprintf(outfp,"P%imax = %.4f     ", i+1, Pi[i][max]);fprintf
(outfp,"\n");
for (i=0; i < NUnit; i++) fprintf(outfp,"P%imin = %.4f     ", i+1, Pi[i][min]);fprintf
(outfp,"\n");
       fprintf(outfp,"RampRate\n");
for (i=0; i < NUnit; i++) fprintf(outfp,"Ur%i = %.2f     ", i+1, Ur[i]);fprintf
(outfp,"\n");
for (i=0; i < NUnit; i++) fprintf(outfp,"Dr%i = %.2f     ", i+1, Dr[i]);fprintf
(outfp,"\n");
       fprintf(outfp,"Hour=0\n");PowerPlant=0;
for (i=0; i < NUnit; i++) fprintf(outfp,"P%i = %.4f     ", i+1, P[i]);fprintf(outfp,"\n");
for (i=0; i < NUnit; i++) PowerPlant=PowerPlant+P[i];
       fprintf(outfp,"PowerPlant = %.4f\n", PowerPlant);
       fprintf(outfp,"Hour\t");
       fprintf(outfp,"PowDemand\t");
       fprintf(outfp,"BestGen\t");
       fprintf(outfp,"PowPlant\t");
        fprintf(outfp,"P_Loss\t");
        fprintf(outfp,"SumPow\t");
       for (i= 0; i < NUnit; i++)   fprintf(outfp,"Pr%imax\t", i+1);
       for (i= 0; i < NUnit; i++)   fprintf(outfp,"Pr%imin\t", i+1);
       fprintf(outfp,"SumPrmax\t"); fprintf(outfp,"SumPrmin\t");
       for (i= 0; i < NUnit; i++)   fprintf(outfp,"P[%i]\t", i+1);
       fprintf(outfp,"Cost\n");
}

void DisplayGA(Word Hour)
{    Word i,j,locate,mainLoop,miniLoop;
     write_xy(2,1,"");
     printf("Hour#     = %5i       SumPr|max = %10.4f   Generation     = %5i\n",Hour+1,
SumPr[max], Generation);
     printf("PowDemand = %10.4f  SumPr|min = %10.4f   BestGeneration= %5i\n",
PowDemand,SumPr[min], BBest.Generation);
     if (NUnit <=10) {    printf("BestChrom = ");
         WriteChrom(BBest.Chrom);printf("\n");
     }

     Decode( BBest.Chrom, P, IC, &PowPlant, &P_Loss, &SumPow);

     Cost  =  CostFunc(P);
     mainLoop = NUnit / 5;
     miniLoop = NUnit % 5;

     printf("#===========#=");
     for (j=1; j <= 3; j++) printf("=============");
     printf("==============#\n");
     for (i=1; i <= 2; i++) {
         for (j=1; j <= 5; j++) {
             locate=5*(i-1) + j;
             switch (locate) {
               case 1  : printf("HBestFitnessH  Pg _%i      ", locate-1); break;
```

```
                case 6  : printf("H  %7.5f  H  Pg _%i      ", BBest.Fitness, locate-1);
break;
                case 5  : printf("Pg _%i  H", locate-1);    break;
               case 10 : printf("Pg _%i  H", locate-1);    break;
                default : printf("Pg _%i      ", locate-1); break;
               }
          }  printf("\n");
     }
     printf("H===========#=");
     for (j=1; j <= 3; j++)  printf("=============");
     printf("=============#\n");

if (NUnit <=60) {
     if (mainLoop > 0) {
          for (i=1; i <= mainLoop; i++) {
               printf("H Pg%2i_|    H", (i-1)/2);
               for (j=1; j <= 4; j++) {
                    locate = 5*(i-1) +j;
                    printf("%9.4f  ", P[locate-1]);
               }
               j=5; locate = 5*(i-1) +j;
               printf("%9.4fH\n", P[locate-1]);
          }
     }

     if ((miniLoop != 0)) {
          i=mainLoop+1;
          printf("H Pg%2i_|    H", (i-1)/2);
          for (j=1; j <= miniLoop; j++) {
              locate = 5*(i-1) +j;
              printf("%9.4f  ", P[locate-1]);
          }
          for (j=1; j <= 5-miniLoop-1; j++) printf("          ");
          printf("         H\n");
     }
}
          printf("#===========#=");
          for (j=1; j <= 3; j++) printf("=============");
          printf("=============#\n");
          printf("HSumPower  =%10.4f MWs          MinCost  =%15.4f BahtH\n",SumPow, Cost);
          printf("HPowerGen  =%10.4f MWs          P_Loss   =%10.4f MWs
H\n",PowPlant,P_Loss);
          printf("#=============");
          for (j=1; j <= 3; j++) printf("=============");
          printf("=============#\n");
          for (j=1; j <= NSpecies; j++) {
               printf("NMicro%i= %4i   ", j, Species[j-1].NMicro);
               if ((NSpecies % 3 == 0) && (j % 3 == 0)) printf("\n");
               if ((NSpecies % 4 == 0) && (j % 4 == 0)) printf("\n");
               else if ((NSpecies % 3 != 0) && (NSpecies % 4 != 0) && (j % 5 == 0)) printf
("\n");
          }
          printf("File Number =%2i", mseed);
         printf("   NMigrate =%4i\n",NMigrate);
          printf("Best FileNumber =%2i\n", BestFile);
          printf("Best Total Cost =%20.4f\n", BestTCost);
          #if    (OS == MS_DOS)
               printf("CPU Time = %3i hours  %2i minutes  %2i seconds  %4i
milli_seconds\n",
                    diff_time.ti_hour, diff_time.ti_min, diff_time.ti_sec,
diff_time.ti_hund*10);
          #elif ((OS == UNIX) || (OS==LINUX))
               printf("CPU Time = %li seconds  %li micro_seconds\n",
                    diff_time.tv_sec, diff_time.tv_usec);
               printf("SumCPU Time = %6i seconds %6i micro_seconds\n",
                    SumTime.tv_sec, SumTime.tv_usec);
          #endif
}

void Report(Word Hour)
{    Word i;
     Check_BestSol();
     ReadTime(start_time,stop_time,&diff_time);
     SumTime.tv_sec += diff_time.tv_sec;
        SumTime.tv_usec += diff_time.tv_usec;
        if(SumTime.tv_usec>=1000000)
        {      do{
```

```
            SumTime.tv_usec -= 1000000;
             SumTime.tv_sec++;
             }while(SumTime.tv_usec >= 1000000);
       }

    DisplayGA(Hour);

    fprintf(outfp,"%i\t",     Hour+1);
    fprintf(outfp,"%8.4f\t", PowDemand);
    fprintf(outfp,"%i\t",     BBest.Generation);
    fprintf(outfp,"%8.4f\t", PowPlant);
    fprintf(outfp,"%8.4f\t", P_Loss);
    fprintf(outfp,"%8.4f\t", SumPow);
    for (i= 0; i < NUnit; i++)     fprintf(outfp,"%8.4f\t",Pr[i][max]);
    for (i= 0; i < NUnit; i++)     fprintf(outfp,"%8.4f\t",Pr[i][min]);
    fprintf(outfp,"%8.4f\t", SumPr[max]);
    fprintf(outfp,"%8.4f\t", SumPr[min]);
    for (i= 0; i < NUnit; i++)     fprintf(outfp,"%8.4f\t", P[i]);
    fprintf(outfp,"%10.5f\t",Cost);
    fprintf(outfp,"%6i\t%6i\n",diff_time.tv_sec,diff_time.tv_usec);

    fflush(outfp);
}


/*-------------------------------------------------------------------------*/
/* utility.c - Timer functions and read data function                      */
/*-------------------------------------------------------------------------*/
void ReadTime(struct timeval str_time,
              struct timeval stp_time,
              struct timeval *diff_time)
{   int cary;
    diff_time->tv_usec = stp_time.tv_usec - str_time.tv_usec;
    if (diff_time->tv_usec < 0)
    {
        diff_time->tv_usec += 1000000; cary = 1;
    }
    else cary = 0;
    diff_time->tv_sec = stp_time.tv_sec - str_time.tv_sec - cary;
    if (diff_time->tv_sec < 0)
    {
        printf("checking time error, because of Y2K problem\n");
        getchar();
    }
    else cary = 0;
}

void Start_Time(void)
{
gettimeofday(&start_time, &timz);
}

void Stop_Time(void)
{
gettimeofday(&stop_time, &timz);
}

void write_xy(int x,int y,char *string)
{   char Command[81];
sprintf(Command,"%s%i;%iH%s\"",echo,x,y,string);
system(Command);
}

void repchar(outfp,Ch,repcount)
FILE *outfp;
char *Ch;
Word repcount;
{   Word j;
    for (j=1; j <= repcount; j++) fprintf(outfp,"%s", Ch);
}

void skip(outfp,skipcount)
FILE *outfp;
Word skipcount;
{   Word j;
    for (j=1; j <= skipcount; j++) fprintf(outfp,"\n");
}
```

```
void Sort_ICLst(iList)
Word *iList;
{    Word i,TempList;
     Flag S;
     for( i=0; i< NUnit; i++) iList[i]=i;
     do { S=0;
          for( i=0; i< NUnit-1; i++)
          {    if ( IC[iList[i]] > IC[iList[i+1]] )
               {      TempList   = iList[i];
                      iList[i]   = iList[i+1];
                      iList[i+1] = TempList;
                               S = 1;
               }
          }
     } while (S!=0);
}

int ReadData()
{   FILE    *Inp , *Outp;
    char    Answer[8];
    Word    i, j, UnitNumber;
    if  (NumFiles < 1)
    {
        printf("Plese enter your Input FileName to solve in ED prog <8 characters>:");
        scanf("%s", InpFileName);
     }
     system(cls);
     strcat(InpFileName,  TypeFileName);
     strcat(OutpFileName, TypeFileName);

/*  Read  File */
    if  ((Inp = fopen(InpFileName, "r"))== NULL)
        {   fprintf(stderr, "Cannot open %s !!\n\a",InpFileName);
            exit (-1);
        }
    printf(" Welcome to Economic Dispatch Program \n");
    printf(" ---------------------------------- \n");
    printf(" Please Enter to load the input data...\n");  /* getchar(); */

    /* Read Data From Input.Dat */
    fscanf(Inp,"%i", &NUnit);
    for (i=0; i < NUnit; i++) {
         fscanf(Inp,"%i", &UnitNumber);
         fscanf(Inp,"%i", &MemberInUnit[i]);
         Member = MemberInUnit[i];
         if  ((Member-1) > 0) {
             for (j=0; j < Member-1; j++) {
                  fscanf(Inp,"%lf", &P_int[i][j]);
                  fscanf(Inp,"%lf", &Ah[i][j]);
                  fscanf(Inp,"%lf", &Bh[i][j]);
                  fscanf(Inp,"%lf", &Ch[i][j]);
             }
             fscanf(Inp,"%lf", &Ah[i][j]);
             fscanf(Inp,"%lf", &Bh[i][j]);
             fscanf(Inp,"%lf", &Ch[i][j]);
         } else {
             fscanf(Inp,"%lf", &Ah[i][0]);
             fscanf(Inp,"%lf", &Bh[i][0]);
             fscanf(Inp,"%lf", &Ch[i][0]);
         }
         fscanf(Inp,"%lf", &Pi[i][max]);
         fscanf(Inp,"%lf", &Pi[i][min]);
         fscanf(Inp,"%lf", &Ur[i]);
         fscanf(Inp,"%lf", &Dr[i]);
         fscanf(Inp,"%lf", &Fuel_Cost[i]);
        }

        /* Start to load demand curve*/
        fscanf(Inp,"%i", &HourDay);
        if (HourDay >= MaxHourDay) {
                    printf("your \"HourDay\" is too large!!\n");
                    printf("Please <ENTER> to exit\n\a");
                    getchar(); exit(-1); }
        /* Initial Power Generation @ Hour=0*/
        Hour=0;
        for (i=0; i < NUnit; i++) fscanf(Inp,"%lf", &Pini[i]);
```

```
        /* Load Demand Curve for Each Hour=1 to HourDay*/
        for (Hour=1; Hour <= HourDay; Hour++)
            fscanf(Inp,"%lf", &PowerDemand[Hour-1]);
        /* Has B-Matrix or not!! */
        fscanf(Inp,"%s",Answer);
        if( strcmp(Answer,"NoLoss") && strcmp(Answer,"NOLOSS") &&
            strcmp(Answer,"noloss"))
             LossFg = LOSS;
        else LossFg = NOLOSS;
        /* >>> B Matrix <<< */
        if (LossFg == LOSS)
        {
           /* Based Power */
              fscanf(Inp,"%lf",&P_Base);
           /* B[0][0]  value  */
              fscanf(Inp,"%lf",&B00);
           /* B[i][0]  vector */
              for (i=0;i<NUnit;i++)  fscanf(Inp,"%lf",&B0[i]);
           /* B[i][j]  matrix */
              for (i=0;i<NUnit;i++)
                  for (j=0;j<NUnit;j++)
                      fscanf(Inp,"%lf",&B[i][j]);
        }
        fclose(Inp);

/* Make Folder Repord */
        sprintf(MyFolder,"Result%i",NUnit);
        sprintf(Command,"mkdir %s",MyFolder);
        system(Command);

/* Make Data Report File -> Output.txt */
        sprintf(OutpFileName,"%s%sOutput.txt",MyFolder,pth);

/* Record File */
        if ((Outp = fopen(OutpFileName, "w"))== NULL)
        {    fprintf(stderr, "Cannot open %s !!\n",OutpFileName);
             exit(-1);
        }
        /* Report the coeficient of system */
        fprintf(Outp,"Number of Units=%i\n", NUnit);
        for (i=0; i < NUnit; i++) {
             fprintf(Outp,"Unit#%i\n", i+1);
             fprintf(Outp,"Member Number of Unit%i = %i\n",i+1,MemberInUnit[i]);
             Member = MemberInUnit[i];
             if  ((Member-1) > 0) {
                 for (j=0; j < Member-1; j++) {
                      if  (j==0)
                           fprintf(Outp,"                    %10.4f MW =<     P[%i]       <
%.4f MW\n",
                                   Pi[i][min], i+1, P_int[i][j]);
                      else
                           fprintf(Outp,"                    %10.4f MW =<     P[%i]       <
%.4f MW\n",
                           P_int[i][j-1], i+1, P_int[i][j]);
                      fprintf(Outp,"H( P[%2i] ) = %15.4f + %10.4f*P[%i] + %10.4f*P
[%i]^2\n",
                              i+1, Ah[i][j], Bh[i][j], i+1, Ch[i][j], i+1);
                 }
                 fprintf(Outp,"                    %10.4f MW =<     P[%i]       =< %.4f
MW\n",
                         P_int[i][j-1], i+1, Pi[i][max]);

                 fprintf(Outp,"H( P[%2i] ) = %15.4f + %10.4f*P[%i] + %10.4f*P[%i]^2\n\n",

                         i+1, Ah[i][j], Bh[i][j], i+1, Ch[i][j], i+1);
             } else {
                 j = 0;
                 fprintf(Outp,"                    %10.4f MW =<     P[%i]       =< %.4f
MW\n",
                         Pi[i][min], i+1, Pi[i][max]);
                 fprintf(Outp,"H( P[%2i] ) = %15.4f + %10.4f*P[%i] + %10.4f*P[%i]^2\n\n",
                         i+1, Ah[i][j], Bh[i][j], i+1, Ch[i][j], i+1);
             }
             fprintf(Outp,"Maximum Power  of unit%i = %15.4f\n",i+1,Pi[i][max]);
             fprintf(Outp,"Minimum Power  of unit%i = %15.4f\n",i+1, Pi[i][min]);
             fprintf(Outp,"Uppr Ramp Rate of unit%i = %15.4f\n",i+1, Ur[i]);
             fprintf(Outp,"Down Ramp Rate of unit%i = %15.4f\n",i+1, Dr[i]);
```

```
            fprintf(Outp,"Fuel Cost of unit%i      = %15.4f\n\n\n",i+1,Fuel_Cost[i]);
        }
        fprintf(Outp,"Start to load demand curve\n");
        fprintf(Outp,"HourDay of load demand curve%i\n", HourDay);
        fprintf(Outp,"Initial Power Generation @ Hour=0\n");
        Hour=0;
        for (i=0; i < NUnit; i++)
             fprintf(Outp,"Unit%i of Hour#%i:  %.4f\n", i+1, Hour, Pini[i]);
        /* Load Demand Curve for Each Hour */
           for (Hour=1; Hour <= HourDay; Hour++)
                fprintf(Outp,"PowerDemand of Hour# %i: %.4f\n",Hour,PowerDemand[Hour-1]);
        /* >>> B Matrix <<< */
           if (LossFg == LOSS)
           {
              /* Based Power */
              fprintf(Outp,"\n\nP_Base = %10.4f\n\n",P_Base);
              /* B[0][0]  value  */
              fprintf(Outp,"B[ 0][ 0] = %11.8f\n\n",B00);
              /* B[i][0]  vector */
              for (i=0;i<NUnit;i++)  fprintf(Outp,"B[%2i][ 0] = %11.8f   ",i+1,B0[i]);
              fprintf(Outp,"\n\n");
              /* B[i][j]  matrix */
              for (i=0;i<NUnit;i++) {
                   for (j=0;j<NUnit;j++)
                        fprintf(Outp,"B[%2i][%2i] = %11.8f   ",i+1,j+1,B[i][j]);
                        fprintf(Outp,"\n");
                }
           }
           fclose(Outp);

/* Display To Check System */
   printf("Unit Number=%d\n", NUnit);
   for (i=0; i < NUnit; i++) {
        system(cls);
        printf("Unit#%i\n", i+1);
        printf("Member Number of Unit%i = %i\n\n", i+1, MemberInUnit[i]);
        Member = MemberInUnit[i];
        if ((Member-1) > 0) {
            for (j=0; j < Member-1; j++) {
                 if (j==0)
                     printf("                    %10.4f MW =<     P[%i]       < %.4f MW\n",
                            Pi[i][min], i+1, P_int[i][j]);
                 else
                     printf("                    %10.4f MW =<     P[%i]       < %.4f MW\n",
                            P_int[i][j-1], i+1, P_int[i][j]);
                 printf("H( P[%2i] ) = %15.4f + %10.4f*P[%i] + %10.4f*P[%i]^2\n",
                        i+1, Ah[i][j], Bh[i][j], i+1, Ch[i][j], i+1);
                 if ( (j+1)%3 == 0 ) printf("\n");
            }
            printf("                  %10.4f MW =<     P[%i]       =< %.4f MW\n",
                   P_int[i][j-1], i+1, Pi[i][max]);
            printf("H( P[%2i] ) = %15.4f + %10.4f*P[%i] + %10.4f*P[%i]^2\n\n",
                   i+1, Ah[i][j], Bh[i][j], i+1, Ch[i][j], i+1);
        } else {
            j = 0;
            printf("                  %10.4f MW =<     P[%i]       =< %.4f MW\n",
                   Pi[i][min], i+1, Pi[i][max]);
            printf("H( P[%2i] ) = %15.4f + %10.4f*P[%i] + %10.4f*P[%i]^2\n\n",
                    i+1, Ah[i][j], Bh[i][j], i+1, Ch[i][j], i+1);
        }
        printf("Maximum  Power of unit%i =%15.4f\n", i+1, Pi[i][max]);
        printf("Minimum  Power of unit%i =%15.4f\n", i+1, Pi[i][min]);
        printf("Uppr Ramp Rate of unit%i =%15.4f\n", i+1, Ur[i]);
        printf("Down Ramp Rate of unit%i =%15.4f\n", i+1, Dr[i]);
        printf("Fuel Cost of unit %i = %15.4f\n", i+1, Fuel_Cost[i]);
        /* getchar(); */
   }

   system(cls);
   printf("Start to load demand curve\n");
   printf("HourDay of load demand curve %i\n", HourDay);
   printf("Initial Power Generation @ Hour=0\n");

   Hour=0;
   for (i=1; i <= NUnit; i++) {
        printf("Unit %i of Hour# %d= %10.4f\n", i, Hour, Pini[i-1]);
        if (i % 10 ==0) printf("\n");
```

```
    }

    /* Load Demand Curve for Each Hour */
    for (Hour=1; Hour <= HourDay; Hour++) {
         printf("PowerDemand of Hour# %d = %10.4f\n", Hour, PowerDemand[Hour-1]);
         if (Hour % 20 ==0) printf("\n");
    }

    /* >>> B Matrix <<< */
    if (LossFg == LOSS)
    {
        /* Based Power */
        printf("\n\nBased Power = %10.4f\n\n",P_Base);
        /* B[0][0]  value  */
        printf("B[ 0][ 0] = %8.5f\n\n",B00);
        /* B[i][0]  vector */
        for (i=0;i<NUnit;i++)  printf("B[%2i][ 0] = %8.5f   ",i+1,B0[i]);
        printf("\n\n");

        /* B[i][j]  matrix */
        for (i=0;i<NUnit;i++) {
             for (j=0;j<NUnit;j++)
                  printf("B[%2i][%2i] = %8.5f   ",i+1,j+1,B[i][j]);
                  printf("\n");
        }
    }

    return 0;
}

/*-------------------------------------------------------------------------*/
/* utriopn.c - contains GA operations                                      */
/*-------------------------------------------------------------------------*/
void Shuffle(Word  *ipick, Population  Pop)
{
     double          temp;
     Word       iother,j;
     Individual    itemp;
     *ipick=0;
     for ( j=0; j < PopSize-1; j++)
     {
         iother = (Byte) rnd(j,PopSize-1);
        itemp  = Pop[iother];
         Pop[iother] = Pop[j];
         Pop[j]    =    itemp;
     }
}

Word TourSelect(Word  PopSize,Population  Pop,Word  *ipick)
{
     Word ifirst,isecond,winner;
     if ((*ipick)+1 > PopSize-1) Shuffle(ipick,Pop);
     ifirst    = (*ipick);
     isecond   = (*ipick)+1;
     (*ipick) += 2;
     if (Pop[ifirst].Fitness > Pop[isecond].Fitness)
             winner = ifirst;
     else    winner = isecond;
     return  winner;
}

Word RoulSelect(Word  PopSize, Population  Pop, double  SumFitness)
{
     Word j;
     double randomperc();
     double Rand,PartSum;

     PartSum = 0.0;  j = -1;
     Rand = SumFitness*randomperc();
     do {   j++;
            PartSum += Pop[j].Fitness;
     } while ((PartSum < Rand) && (j<PopSize));
     return j;
}

void Mutation( Chromosome  Child, double  *PMutation, Word *NMutation)
{
```

```
    Word   i, j, mask, temp=1;

    for ( i=0; i < NGenes; i++)
    {   mask = 0;
        for ( j=0; j < ParaBit; j++)
        {
            if (flip(*PMutation))
            {
                mask |= (temp<<j);
                (*NMutation)++;
            }

       }   Child[i] ^= mask;
    }
}

void UniCrossover(Chromosome  Parent1,Chromosome  Parent2,
                  Chromosome  Child1, Chromosome  Child2,
                  double  *PCross,
                  Word    *NCross)
{
    Word j, k, xcross,temp=1;

    for ( j=0; j < NGenes; j++) {
          xcross=0;
          for ( k=0; k < ParaBit; k++) {
                xcross <<= 1;
                if ( flip(*PCross) )
                {    xcross |= temp;
                     (*NCross)++;
                }
          }
          Child1[j]=(Parent1[j]&(~xcross)) | (Parent2[j]&  xcross);
          Child2[j]=(Parent1[j]&  xcross)  | (Parent2[j]&(~xcross));
    }
```