```
//              ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//              ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Com 28       Controlling Object For The Allocation Function Of Strategic Weapon
//              Systems Against Scenario Specific targetss
//
//              Note:  must be able to handle multiple scenarios
//              ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//              ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

   class Allocation

  {
   public:
//
//              Define addiitonal model parameters:
//
   int exists;
   char* obj_name;

   long RN_seed;
   int DEBUG, fit_func;
   float fuzzy_level;

   int max_targets ;
   int base_line_WH, new_WH, total_WH ;
   int total_inventory, no_option;
   int Restart_iter, ga_iter, ga_pop;
   float pd_objective, F1_form_factor, beta[5];
   float mutation_pr, xchange_pr;

   WEAPON_ *sys;
   Mission_ startX;
   GA_Alloc soln;
   int **opt_index;

   Allocation();                          // Constructor
//
//              overload the input and output operators
//
   friend istream& operator>>( istream& in,  Allocation&  );
   friend ostream& operator<<( ostream& out, Allocation&  );
//
//              Define addiitonal member functions:
//
   float Allocation_Opt( );               // Opt. The Allocation Of Weapons To Targets
   float Allo_Fitness( int*, int*, float* );// Controller For Total Fitness Assessment
```

```
float Fitness_PD(int, float*, float, int );// Goal Pd Fitness Function
float Fitness_Stk_Lmt( int );        // Stockpile Limit Fitness Function
 float Fitness_Time( int );          // Time Urgency Fitness Function
float Fitness_Wt( int );             // Target Weighting Fitness Function
float Fitness_Yield( float );        // Minimum Yield Fitness Function
void Genetic_Setup( );
void IC_Alloc( );
void Perf_Results( int );
float Pssk( int, float, float );     // Determine the probability of "single shot" kill
float Obj_Pk( int, float, float );
//
//              Define a dump and restart capability
//
  void Gen_Dump( ostream& );
  void Gen_Restart( istream& );
 };    // Block: 59
```

Allocation::Allocation( )

```
 {
  obj_name = new char[10];
//
//              Define default conditions
//
  exists=FALSE;
  strcpy( obj_name,"None" );
  RN_seed=1111111111;
  DEBUG=FALSE;
  fit_func=1;
  fuzzy_level=1.25;

  Restart_iter=0;
  ga_iter=75;
  ga_pop=75;

  max_targets=1;
  base_line_WH=1;
  new_WH=0 ;
  total_WH= base_line_WH+new_WH ;
  no_option=0;

  pd_objective=0.75;
  F1_form_factor=0.025;
  mutation_pr=0.05;
  xchange_pr=0.45;
//
```

```
// Com 29      Beta Is The Decision Model Weighting Coeficients.
//
   for( int i=0; i<5; i++ )
     beta[i]=1.0;
  } ;     // Block: 60
//             ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

    istream& operator>>( istream& in, Allocation& adum )

  {
   char tmp_in[15];
   char tmp_file[15];
   adum.exists=TRUE;
   int tmp;
   in >> adum.obj_name;
   in >> tmp_in;
   while( strcmp( tmp_in,"eoi" ) !=0 )
  {
     if( strcmp( tmp_in,"Restart" ) == 0 )
       {
       in >> tmp_file ;
       ifstream Restart_file( tmp_file ) ;
       Restart_file >> adum.Restart_iter >> adum.RN_seed;
       adum.Gen_Restart( Restart_file ) ;
       }
     else if( strcmp( tmp_in,"Targets" ) == 0 )
       {
       in >> adum.startX;
       adum.max_targets = adum.startX.max_targets ;
       }
     else if( strcmp( tmp_in,"Weapons" ) == 0 )
       {
//
//             First load baselines and other allocation parameters
//             Next load the CEP values and associated stockpile assignments
//
       in >> adum.base_line_WH >> adum.new_WH;
       adum.total_WH = adum.base_line_WH + adum.new_WH;
       adum.sys= new WEAPON_[ adum.total_WH+1 ];
       for( int k=1; k<= adum.total_WH; k++ )
         in >> adum.sys[ k ] ;
       }
    else if( strcmp( tmp_in,"Population" ) == 0 )
      in >> adum.ga_pop;
    else if( strcmp( tmp_in,"Convergence" ) == 0 )
      in >> adum.ga_iter;
```

```
        else if( strcmp( tmp_in,"Mutation" ) == 0 )
          in >> adum.mutation_pr;
        else if( strcmp( tmp_in,"Xchange" ) == 0 )
          in >> adum.xchange_pr;
        else if( strcmp( tmp_in,"Debug" ) == 0 )
          {
          in >> adum.fit_func;
          adum.DEBUG = TRUE ;
          }
        else if( strcmp( tmp_in,"ObjectivePD" ) == 0 )
//
//                  Objective Mission Performance
//
          in >> adum.pd_objective;
        else if( strcmp( tmp_in,"PD_fitness" ) == 0 )
//
//                  Factor Defining The Degree Of Acceptance Of The Objective Pd
//
          in >> adum.F1_form_factor;
        else
    {
          cerr << "Error: (Allocation Input) unaceptable input option : " << tmp_in
          << "\n" << flush;
    } ;     // Block: 61
    in >> tmp_in;
    } ;     // Block: 62
//
//                  Need To Initialize The Genetic Chromosome
//
    adum.Genetic_Setup();
    return in;
    } ;     // Block: 63
//                  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

ostream& operator<<(ostream& out, Allocation& A)

```
    {
//
//                  provide coding to represent the ouput desired for this object
//
    out << "\n\n" << A.obj_name << "\n";

    out << "Objective Pd  " <<  A.pd_objective << "\n";
    out << "Targets " << A.max_targets << "\n";
    out << "Weapons " << A.total_WH << "\n";
    for(  int k=1; k<= A.total_WH; k++ )
```

```
    out << A.sys[k] <<" "<< "\n";
  out << "Population  " << A.ga_pop << "\n";
  out << "Convergence  " << A.ga_iter << "\n";
  out << "Mutation Pr  " << A.mutation_pr << "\n";
  out << "Xover Pr  " << A.xchange_pr << "\n";
  return out;
};     // Block: 64
//          ++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

float Allocation::Allocation_Opt( )

```
{
 float *fit;
 float *raw_fit;
 fit = new float[ ga_pop ];
 raw_fit = new float[ ga_pop ];
 float debug_fitness;
 ofstream debug_out("ZALLO_debug.dat",ios::app);

 if( DEBUG )
   debug_out << "\n" <<
   "++++++++++ Allocation Fitness Debug ++++++++++++++++++++++ \n";

 IC_Alloc( );

 float **suitability;
 suitability = new float*[ ga_pop ] ;
 for( int su=0; su<ga_pop; su++ )
   suitability[su] = new float[ max_targets ] ;

 for( int iter=Restart_iter; iter<ga_iter; iter++ )
   {
//
// Com 30      Setup And Calculate The Fitness For Each Member Of The
//             Chromosome Population
//
    for( int k=0; k<soln.ga_pop; k++ )
      fit[ k ] = Allo_Fitness(  soln.ga_chromo[ k ], opt_index[k], suitability[k] );
//
//              Dump Info For Restart Or Debug
//
    ofstream dump_out("ZALLO_dump.dat");
    RN_seed = soln.random.get_seed();
    dump_out << iter << " " << RN_seed << "\n";
    Gen_Dump( dump_out );
    if( DEBUG )
```

```
            {
            debug_out << "Fitness Results, all Chromosomes: ( iter "<<iter<<" )\n";
            for( int ka=0; ka<soln.ga_pop; ka++ )
               debug_out<< fit[ ka ] << " ";
            }
//
// Com 31      Transfer Fitness To GA Algorithms
//
      soln.GA_Alloc_Fitness( fit );
//
//              Setup For A Debug Calculation And Output
//
      if( DEBUG )
         {
         debug_fitness= Allo_Fitness (  soln.ga_chromo[ soln.opt_soln ],
                     opt_index[soln.opt_soln], suitability[ soln.opt_soln ] );
         debug_out << "Iter : "<<iter<<"  Fitness = "<<debug_fitness
         <<" ( func no. "<<fit_func<<" ) \nAllocation Vector \n" ;
         for( int kb=0; kb<max_targets; kb++ )
            debug_out << soln.ga_chromo[soln.opt_soln][kb] << " "
            << opt_index[soln.opt_soln][kb] << "   ";
         debug_out << "\n";
         }
//
// Com 32      Create The Next Generation
//
      soln.Next_Gen_GA_Alloc( suitability, fuzzy_level );
      }
   if( DEBUG )
      debug_out << "\n" <<
      "++++++++++ Allocation Fitness Debug End ++++++++++++++++++++++ \n";
//
//              Output The Results Of The Allocation
//
   int k_soln=0;
   Perf_Results( k_soln );

   return( fit[ soln.opt_soln ] ) ;
  };      // Block: 65
//          ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

float Allocation::Allo_Fitness(int *ga_soln, int *cep_index, float *suit)

```
  {
//
```

```
//              This routine provides the decision model, currently trivial, for
//              assessing the various goals and objectives of an allocation
//              algorithm.
//
   int jj, sys_index, sys_inv;
   float yld;
   float *cep;
//
//              Determin the total allocation of weapons for  a chromosome
//
   for( int k=0; k<= total_WH; k++ )
     sys[ k ].allocated =0;
//
// Com 33       Note: ga_soln Provides The Mapping Between Targets And Options
//              assign Provides The Mapping Between Options And Inventories
//
   for( k=0; k< max_targets; k++ )
     {
     jj = ga_soln[ k ] ;
     sys[ jj ].allocated +=1;
     }
//
//              Assess Fitness For Each Decision Vriable, ie. F1-F5
//              Assume equal importance for this version: 11/20/97
//
   int indx, i2 ;
   float total_fitness;
   for( k=0; k<max_targets; k++ )
     {
     sys_index = ga_soln[k];
//
//              Note: Fitness Deined As Weighted Sum Of Obj Function;
//              Product For All Targets  PI( (1+sum(fi))j )
//
     total_fitness=0.0;
     if( sys_index != no_option )
       {
       cep= sys[ sys_index ].ceps ;
       yld= sys[ sys_index ].yield ;
       i2= sys[ sys_index ].total_fz_opt ;
       indx = Fitness_PD( k, cep,yld, i2 );
       cep_index[ k ] = indx;
       total_fitness += beta[0]*Obj_Pk( k, cep[indx],yld ) ;
       total_fitness += beta[1]*Fitness_Yield( yld );
//              total_fitness += beta[2]*Fitness_Stk_Lmt( k );
//              total_fitness += beta[3]*Fitness_Wt( k );
```

```
//              total_fitness += beta[4]*Fitness_Time( k );
        }
      suit[k]=total_fitness ;
      }
//
//              Fitness For The Alocation Is The Percent Of Acceptable
//              Individual Targetings
//
  total_fitness=0.0;
  for( k=0; k<max_targets; k++ )
    if( suit[k] >= fuzzy_level )
      total_fitness +=1.0 ;
  total_fitness /= float( max_targets );

  return( total_fitness );
 };     // Block: 66
//          +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

float Allocation::Fitness_PD(int targ_indx, float *cep, float yld, int fz_opts)

```
 {
//
//              This routine provides the fitness based on goal or mission targets
//              damage expectancy.  Too little or too much is not a good solution.
//
  float Pk, F1 ;
  int indx=0;
  float max_pk=0.0;
  for( int k=0; k< fz_opts; k++ )
    {
    Pk = Pssk( targ_indx, cep[k], yld );
    F1 = exp(-(Pk-pd_objective)*(Pk-pd_objective)/F1_form_factor );
    if( F1 > max_pk )
      {
      max_pk=F1;
      indx= k;
      }
    };
  return( indx );
 };     // Block: 67
//          +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

float Allocation::Fitness_Stk_Lmt(int inv_num)

```
 {
```

```
//
//              This routine is a fitness representations for stockpile constraints;
//              not a good idea to allocate weapons you do not have.
//
   float F3=1;
   int num_avail, num_alloc ;
   num_avail= sys[ inv_num ].inventory;
   num_alloc= sys[ inv_num ].allocated;
   F3 = 1.0/( 1.0+exp(num_alloc-num_avail) );
   return( F3 );
  };    // Block: 68
//          ++++++++++++++++++++++++++++++++++++++++++++++++++++
```

float Allocation::Fitness_Time(int targ_indx)

```
  {
//
//              This routine captures fitness for a time urgency metric; ie.
//              reconstitution targets, time urgent targets, and those of uniform
//              importance as functions of time.
//
   float F5=1.0;
   return( F5 );
  };    // Block: 69
//          ++++++++++++++++++++++++++++++++++++++++++++++++++++
```

float Allocation::Fitness_Wt(int targ_indx)

```
  {
//
//              This routine is the fitness correlation based on target
//              importance factors.
//
   float F4=1;
   F4 = startX.targs[ targ_indx ].Wt ;
   return( F4 );
  };    // Block: 70

//          ++++++++++++++++++++++++++++++++++++++++++++++++++++
```

float Allocation::Fitness_Yield(float yld)

```
  {
//
//              This routine provides a rule to minimize yield applied to a target.
//              Note:  This Routine Is Valid If The Yield Options
//                   Are Not Less Than 100
```

```
//
   float min_yield=100.0 ;
   float F2=1;
   F2 = pow( (min_yield/yld),0.3333);
   return( F2 );
 };      // Block: 71
//          +++++++++++++++++++++++++++++++++++++++++++++++++++++

   void Allocation::Genetic_Setup()

 {
//
// Com 34       Pass Parameters Read Into The Allocation Class To Initialize
//              The Genetic algorithms.
//
   soln.selections = total_WH ;
   soln.max_gene = max_targets;
   soln.ga_pop = ga_pop;

   opt_index=new int*[ ga_pop ];
   for( int i=0; i<ga_pop; i++ )
   opt_index[i]= new int[ max_targets ];

   soln.mutation_pr = mutation_pr;
   soln.xchange_pr = xchange_pr;
//
// Com 35       Initialize Space For Chromosomes In The Ga
//
   soln.Init_GA_Alloc( );
//
//              Identify Total Warhead Inventories
//
   int total_stockpile=0 ;
   soln.random.seed( RN_seed );
   for( int k=1; k<=total_WH; k++ )
     total_stockpile += sys[k].inventory ;
   total_inventory= total_stockpile;

   return ;
 };      // Block: 72
//          +++++++++++++++++++++++++++++++++++++++++++++++++++++

   void Allocation::IC_Alloc( )

 {
//
```

```
//              This Routine Provides GA Setups For The Allocation Problem.
//              Normal GA Setup Is Handled By The Genetic Setup
//              Routine:  Init_GA_Alloc().
//
   int k_option;

   float pr_alloc, pr_tmp;
   pr_alloc = float( total_inventory/max_targets ) ;
   if( pr_alloc >=1.0 )
     pr_alloc= 1.0 ;

   float *tmp_hist;
   tmp_hist= new float[ total_inventory ];
   for( int i=0; i<soln.ga_pop; i++ )
     {
//
//              Create An Experimental Distribution For Defining Initial Allocations
//
   int indx=0;
   for( int m=1; m<=total_WH; m++ )
     for( int n=0; n<sys[ m ].inventory; n++ )
       {
       tmp_hist[indx]=m;
       indx ++;
       }
   soln.random.Setup_exp_dist( tmp_hist, total_inventory );
//
//              Set The Initial Allocations
//
     for( int k=0; k<soln.max_gene; k++ )
       {
       pr_tmp= soln.random.fdraw();
       if( pr_tmp <= pr_alloc )
         {
         k_option= int( soln.random.exp_dist( ) ) ;
         soln.ga_chromo[i][k] = k_option;
         }
       else
         soln.ga_chromo[i][k] = no_option;
       }
     }

  return ;
 } ;     // Block: 73
//          +++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

void Allocation::Perf_Results(int soln_index)

```
{
 ofstream Monitor_out("Zresults.dat",ios::app);
 int sys_index, indx1, avn, ak ;
 char att;
 float Pk, yld, tmp_fl ;
 float cep ;
 Monitor_out.precision(6);
 float *tmp_suit ;
 tmp_suit= new float[ max_targets ] ;
//
//                Create A Database Of Mission Performance And Allocation
//
 tmp_fl = Allo_Fitness(  soln.ga_chromo[ soln_index ],
     opt_index[soln_index], tmp_suit );

 Monitor_out << "\nPerformance results: \n"
 << "   :: vn ::  :: t ::  :: k ::  :: index ::  :: yield ::  :: cep ::  :: Pk ::\n";
 for( int k=0; k<max_targets; k++ )
   {
   sys_index = soln.ga_chromo[ soln_index ][k];
   avn= startX.targs[ k ].vn;
   att= startX.targs[ k ].tc;
   ak= startX.targs[ k ].k;
   if( sys_index > 0 )
     {
     indx1 = opt_index[ soln_index ][k];
     cep= sys[ sys_index ].ceps[ indx1 ];
     yld= sys[ sys_index ].yield;
     Pk =Pssk( k, cep, yld );
     }
   else
     {
     Pk=0.0;
     cep=1e10;
     }
   Monitor_out << avn << " \t " << att << " \t " << ak << " \t " << sys_index
   << " \t " << yld << " \t " << cep << " \t " << Pk << "\n";
   }
//
//                Create A Series Of Allocaation Databases
//
 for( int ka=1; ka<=total_WH; ka++ )
   {
   Monitor_out << "\nTarget Allocations : ( inv no. " << ka << " )\n";
```

```
      for( int kb=0; kb<max_targets; kb++ )
        {
        sys_index = soln.ga_chromo[ soln_index ][kb];
        indx1 = opt_index[ soln_index ][kb];
        if( sys_index == ka )
          {
          cep= sys[ sys_index ].ceps[indx1];
          yld= sys[ sys_index ].yield;
          avn= startX.targs[ kb ].vn;
          ak= startX.targs[ kb ].k;
          Monitor_out << avn << " \t " << ak << " \t " << sys_index << " \t " << yld
          << " \t " << cep << "\n";
          }
        }
      }
//
//              Print Out Allocated Inventories
//
  Monitor_out << "\nWeapon System Inventories \n";
  for( k=0; k<= total_WH; k++ )
    Monitor_out << k << " \t " << sys[ k ].inventory << " \t " << sys[ k ].allocated
    << "\n";
  return ;
 } ;     // Block: 74
//           ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

float Allocation::Pssk(int targ_indx, float cep, float yld)

```
 {
  float Pk =1.0 ;
  float a;
  int avn,at,ak;
  char atc;
//
//              Define The Weapon Radius For The Target
//
  avn= startX.targs[ targ_indx ].vn;
  atc = startX.targs[ targ_indx ].tc;
  ak = startX.targs[ targ_indx ].k;

  startX.Vul_Setup( avn,atc,ak, yld ) ;
//
//              Use A Simple WR/Cep Correlation For Kill Probability
//
  a = (startX.wr0/cep)*(startX.wr0/cep);
  Pk = 1.0 - pow( 0.5,a );
```

```
   return( Pk );
  } ;      // Block: 75
//           ++++++++++++++++++++++++++++++++++++++++++++++++++++

    float Allocation::Obj_Pk( int targ_indx, float cep, float yld )

  {
   float Pk =1.0 ;
   float a, F1;
   int avn,at,ak;
   char atc;
//
//              Define The Weapon Radius For The Target
//
   avn= startX.targs[ targ_indx ].vn;
   atc = startX.targs[ targ_indx ].tc;
   ak = startX.targs[ targ_indx ].k;

   startX.Vul_Setup( avn,atc,ak, yld ) ;
//
//              Use A Simple WR/Cep Correlation For Kill Probability
//
   a = (startX.wr0/cep)*(startX.wr0/cep);
   Pk = 1.0 - pow( 0.5,a );

   F1 = exp(-(Pk-pd_objective)*(Pk-pd_objective)/F1_form_factor );
   return( F1 );
  } ;      // Block: 76
//           ++++++++++++++++++++++++++++++++++++++++++++++++++++

    void Allocation::Gen_Dump(  ostream& out )

  {
//
//              Dump All Information Related To Allocation To A File For Restart
//
   int k;
   out << exists << “\n” ;
   out << DEBUG << “ “ << fit_func << “\n” ;
   out << max_targets << “\n” ;
   out << base_line_WH << “ “ << new_WH << “ “ << total_WH << “\n”  ;
   out << ga_iter << “ “ << ga_pop << “\n” ;
   out << pd_objective << “ “ << F1_form_factor << “ “ << beta[0] << “ “
   << beta[1] << “ “ << beta[2] << “ “ << beta[3] << “ “ << beta[4] << “\n” ;
   out << mutation_pr << “ “ << xchange_pr << “\n” ;
   for ( k=0; k<= total_WH; k++ )
```

```
      out << sys[k] << "\n" ;
    out << "\n" ;
    out << startX.max_targets << "\n" ;
    for( k=0; k<startX.max_targets; k++ )
      out << startX.targs[k];
    out << "\n" ;
    out << soln.selections << " " << soln.max_gene << " " << soln.ga_pop << " "
    << soln.mutation_pr << " " << soln.xchange_pr << " " << soln.opt_soln << "\n";
    for( k=0; k<soln.ga_pop; k++ )
      {
      for( int kk=0; kk<soln.max_gene;kk++ )
        out << soln.ga_chromo[k][kk] << " "<< opt_index[k][kk] << "   " ;
      out << "\n" ;
      }
    out << "\n" ;
    return;
  } ;      // Block: 77
//         ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

void Allocation::Gen_Restart(istream& in)

```
  {
//
//              Read All Information Related To Allocation To A File For Restart
//
    in >> exists ;
    in >> DEBUG >> fit_func ;
    in >> max_targets ;
    in >> base_line_WH >> new_WH >> total_WH ;
    sys= new WEAPON_[ total_WH+1 ];
    for( int k=0; k<= total_WH; k++ )
      in >> sys[k] ;
    in >> ga_iter >> ga_pop ;
    in >> pd_objective >> F1_form_factor >> beta[0] >> beta[1] >> beta[2] >>
     beta[3] >> beta[4] ;
    in >> mutation_pr >> xchange_pr ;
    in >> startX.max_targets;
      startX.targs= new TARGET_[ startX.max_targets ] ;
    for( k=0; k<startX.max_targets; k++ )
      startX.targs[k].Reset_Targ( in ) ;
    in >> soln.selections >> soln.max_gene >> soln.ga_pop >> soln.mutation_pr
     >> soln.xchange_pr >> soln.opt_soln ;
      soln.Init_GA_Alloc();
    for( k=0; k<soln.ga_pop; k++ )
      for( int kk=0; kk<soln.max_gene;kk++ )
        in >> soln.ga_chromo[k][kk] >> opt_index[k][kk] ;
```

```
return;
};      // Block: 78
```