7.16 Anexo 16 Código del programa realizado

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

#define NUMCOMBINACIONES 50 /* Numero de combinaciones de la poblacion */
#define T 100


/* Definicion de la estructura para componentes redundantes y equipos de
mantenimiento */
int CRedundantes[NUMCOMBINACIONES+2][5];
int Eq[NUMCOMBINACIONES+2][6]; /* la posicion 5 del arreglo mantiene la suma de los
Eq[i] */
int FactorF[NUMCOMBINACIONES+2][5];
double Confiabilidad[NUMCOMBINACIONES+2][5];
double MTTR1[NUMCOMBINACIONES+2][5];
double Disponibilidad[NUMCOMBINACIONES+2][6]; /* la posicion 5 del arreglo mantiene
la disp. total */
int Datos[5][7]; /* Datos leidos desde el archivo */
double Costos[NUMCOMBINACIONES+2][5]; /* 0-costo del diseño, 1-peso, 2-volumen,
3-mantenimiento, 4-CostoTotal */
double Cos[NUMCOMBINACIONES][2];
double Dis[NUMCOMBINACIONES][2];

void GenerarIndiceOrdenamiento();

/*********************************************************/
/*** FUNCIONES DE APOYO PARA EL CALCULO DE LAS FUNCIONES ***/
/*********************************************************/

/* Obtiene un numero aleatorio entre 1 y bound */
int aleatorio(int bound) {
  int valor;

  valor = (((bound)*rand()) / RAND_MAX) + 1;
  if (valor > bound) return(bound);
  else if (valor < 1) return(1);
  else return(valor);
}

/* funcion pow: eleva el valor 'base' a la 'potencia' potencia */
double elevar(double base, int potencia) {
  double res;
  int veces;

  res = base;
```

```
  for (veces = 0; veces < potencia-1; veces++) {
    res = res * base;
  }
  return res;
}
/* Intercambia dos valores doubles */
void Swap(double *A, double *B) {
  double tmp;

  tmp = *A;
  *A = *B;
  *B = tmp;
}

/* Intercambia dos valores enteros */
void SwapInt(int *A, int *B) {
  int tmp;

  tmp = *A;
  *A = *B;
  *B = tmp;
}

/* Inicializacion del generador de numeros aleatorios */
void Inicializa() {
  srand( (unsigned)time( NULL ) );
}

/*****************************************************************************/
/********** FUNCIONES DE IMPRESION PARA EL SISTEMA **********/
/*****************************************************************************/

int posCosto(double combinacion) {
  int i;

  for (i=0; i<NUMCOMBINACIONES; i++) {
    if (Cos[i][0] == combinacion) return(i);
  }
  return(0);
}

int posDisponibilidad(double combinacion) {
  int i;

  for (i=0; i<NUMCOMBINACIONES; i++) {
    if (Dis[i][0] == combinacion) return(i);
  }
  return(0);
}
```

```
/* Imprime los resultados a un archivo de texto llamado: result.txt */
void imprimeresultadoarchivo(char archivo[15]) {
  int i, j;
  FILE *fp;

  if ((fp = fopen(archivo, "w")) != NULL) {
    fprintf(fp,
"=====================================================================
=\n");
    fprintf(fp, "Indice\tCRedun\t\tEqi\t\tAs\tCT\n");
    fprintf(fp,
"=====================================================================
=\n");
    for (i=0; i<NUMCOMBINACIONES; i++) {
                          fprintf(fp, "%d\t", i);
      for (j=0; j<5; j++) {
                           fprintf(fp, "%d ", CRedundantes[i][j]);
                          }
                          fprintf(fp, "\t");
                          for (j=0; j<5; j++) {
                           fprintf(fp, "%d ", Eq[i][j]);
                          }
                          fprintf(fp, "\t%d", posDisponibilidad(i));
                          fprintf(fp, "\t");
                          fprintf(fp, "%2.2f%", Disponibilidad[i][5]*100);
                          fprintf(fp, "\t%d", posCosto(i));
                          fprintf(fp, "\t");
                          fprintf(fp, "%2.2f\n", Costos[i][4]);
                          fprintf(fp, "---------------------------------------------------------------------\n");
                         }
  }
}

/* Imprime los resultados a pantalla y de ser posible a un archivo de texto */
void ImprimePoblacion() {
  int i, j;

  GenerarIndiceOrdenamiento();
  printf("=============================================================\n");
  printf("Indice\tCRedun\t\tEqi\t\tRAs\t\tAs\tRCT\tCT\n");
  printf("=============================================================\n");
  for (i=0; i<NUMCOMBINACIONES; i++) {
                         printf("%d\t", i);
    for (j=0; j<5; j++) {
                          printf("%d ", CRedundantes[i][j]);
                         }
                         printf("\t");
                         for (j=0; j<5; j++) {
                          printf("%d ", Eq[i][j]);
```

```
                        }
                        printf("\t%d", posDisponibilidad(i));
                        printf("\t");
                        printf("%2.2f%", Disponibilidad[i][5]*100);
                        printf("\t%d", posCosto(i));
                        printf("\t");
                        printf("%2.2f\n", Costos[i][4]);
                        printf("-----------------------------------------------------------------\n");
  }
  imprimeresultadoarchivo("pobinicial.txt");
}

/* Imprime los resultados a pantalla y de ser posible a un archivo de texto */
void ImprimirResultados() {
  int i, j;
  int combi[10];
  double costo[10], disp[10];


printf("=================================================================\n");
  printf("Indice\tCRedun\t\tEqi\t\tRAs\t\tAs\tRCT\tCT\n");

printf("=================================================================\n");
  for (i=0; i<NUMCOMBINACIONES; i++) {
                        printf("%d\t", i);
    for (j=0; j<5; j++) {
                          printf("%d ", CRedundantes[i][j]);
                        }
                        printf("\t");
                        for (j=0; j<5; j++) {
                          printf("%d ", Eq[i][j]);
                        }
                        printf("\t%d", posDisponibilidad(i));
                        printf("\t");
                        printf("%2.2f%", Disponibilidad[i][5]*100);
                        printf("\t%d", posCosto(i));
                        printf("\t");
                        printf("%2.2f\n", Costos[i][4]);
                        printf("-----------------------------------------------------------------\n");
  }
  imprimeresultadoarchivo("pobfinal.txt");

  /* busca los mejores valores */
  costo[0] = costo[1] = costo[2] = costo[3] =  costo[4] = costo[5] = costo[6] = costo[7] =
costo[8] =  costo[9] = 99999;
  for (i=0; i<NUMCOMBINACIONES; i++) {
                          /* entre 100 y 98 */
                          if ((Dis[i][1] <= 1.00) && (Dis[i][1]) >= .98) {
                                        if (costo[0] > Costos[(int)Dis[i][0]][4]) {
```

```
                                        costo[0] = Costos[(int)Dis[i][0]][4];
                                        combi[0] = (int)Dis[i][0];
                                        disp[0] = Dis[i][1];
                        }
}
 /* entre 98 y 96 */
if ((Dis[i][1] <= .98) && (Dis[i][1]) >= .96) {
                        if (costo[1] > Costos[(int)Dis[i][0]][4]) {
                                        costo[1] = Costos[(int)Dis[i][0]][4];
                                        combi[1] = (int)Dis[i][0];
                                        disp[1] = Dis[i][1];
                        }
}
 /* entre 96 y 94 */
if ((Dis[i][1] <= .96) && (Dis[i][1]) >= .94) {
                        if (costo[2] > Costos[(int)Dis[i][0]][4]) {
                                        costo[2] = Costos[(int)Dis[i][0]][4];
                                        combi[2] = (int)Dis[i][0];
                                        disp[2] = Dis[i][1];
                        }
}
 /* entre 94 y 92 */
if ((Dis[i][1] <= .94) && (Dis[i][1]) >= .92) {
                        if (costo[3] > Costos[(int)Dis[i][0]][4]) {
                                        costo[3] = Costos[(int)Dis[i][0]][4];
                                        combi[3] = (int)Dis[i][0];
                                        disp[3] = Dis[i][1];
                        }
}
 /* entre 92 y 90 */
if ((Dis[i][1] <= .92) && (Dis[i][1]) >= .90) {
                        if (costo[4] > Costos[(int)Dis[i][0]][4]) {
                                        costo[4] = Costos[(int)Dis[i][0]][4];
                                        combi[4] = (int)Dis[i][0];
                                        disp[4] = Dis[i][1];
                        }
}
  /* entre 90 y 88 */
if ((Dis[i][1] <= .90) && (Dis[i][1]) >= .88) {
                        if (costo[5] > Costos[(int)Dis[i][0]][4]) {
                                        costo[5] = Costos[(int)Dis[i][0]][4];
                                        combi[5] = (int)Dis[i][0];
                                        disp[5] = Dis[i][1];
                        }
}
  /* entre 88 y 86 */
if ((Dis[i][1] <= .88) && (Dis[i][1]) >= .86) {
                        if (costo[6] > Costos[(int)Dis[i][0]][4]) {
                                        costo[6] = Costos[(int)Dis[i][0]][4];
```

```
                                                combi[6] = (int)Dis[i][0];
                                                disp[6] = Dis[i][1];
                                        }
                                }
                                  /* entre 86 y 84 */
                                if ((Dis[i][1] <= .86) && (Dis[i][1]) >= .84) {
                                        if (costo[7] > Costos[(int)Dis[i][0]][4]) {
                                                costo[7] = Costos[(int)Dis[i][0]][4];
                                                combi[7] = (int)Dis[i][0];
                                                disp[7] = Dis[i][1];
                                        }
                                }
                                  /* entre 84 y 82 */
                                if ((Dis[i][1] <= .84) && (Dis[i][1]) >= .82) {
                                        if (costo[8] > Costos[(int)Dis[i][0]][4]) {
                                                costo[8] = Costos[(int)Dis[i][0]][4];
                                                combi[8] = (int)Dis[i][0];
                                                disp[8] = Dis[i][1];
                                        }
                                }
                                  /* entre 82 y 80 */
                                if ((Dis[i][1] <= .82) && (Dis[i][1]) >= .80) {
                                        if (costo[9] > Costos[(int)Dis[i][0]][4]) {
                                                costo[9] = Costos[(int)Dis[i][0]][4];
                                                combi[9] = (int)Dis[i][0];
                                                disp[9] = Dis[i][1];
                                        }
                                }

  }

  printf("\n++++++++++++++++++++++++++++++\n");
  printf("Los Mejores Resultados son: \n");
  printf("++++++++++++++++++++++++++++++\n");
  //printf("Disp: [100-95]: %d %2.2f %2.2f\n", combi[0], disp[0]*100, costo[0]);

  if (costo[0] != 99999) {
                        printf("Disp: [100-98]: %d\t%2.2f %2.2f\t", combi[0], disp[0]*100,
costo[0]);
                        for (j=0; j<5; j++) printf("%d ", CRedundantes[combi[0]][j]);
                        printf("\t");
                        for (j=0; j<5; j++) printf("%d ", Eq[combi[0]][j]);
                        printf("\n");
  } else printf("Disp: [100-98]: No existe combinacion en este rango!\n");

  if (costo[1] != 99999) {
                        printf("Disp:  [98-96]: %d\t%2.2f %2.2f\t", combi[1], disp[1]*100,
costo[1]);
                        for (j=0; j<5; j++) printf("%d ", CRedundantes[combi[1]][j]);
```

```
                        printf("\t");
                        for (j=0; j<5; j++) printf("%d ", Eq[combi[1]][j]);
                        printf("\n");
  } else printf("Disp: [98-96]: No existe combinacion en este rango!\n");

  if (costo[2] != 99999) {
                        printf("Disp:  [96-94]: %d\t%2.2f %2.2f\t", combi[2], disp[2]*100,
costo[2]);
                        for (j=0; j<5; j++) printf("%d ", CRedundantes[combi[2]][j]);
                        printf("\t");
                        for (j=0; j<5; j++) printf("%d ", Eq[combi[2]][j]);
                        printf("\n");
  } else printf("Disp: [96-94]: No existe combinacion en este rango!\n");

  if (costo[3] != 99999) {
                        printf("Disp:  [94-92]: %d\t%2.2f %2.2f\t", combi[3], disp[3]*100,
costo[3]);
                        for (j=0; j<5; j++) printf("%d ", CRedundantes[combi[3]][j]);
                        printf("\t");
                        for (j=0; j<5; j++) printf("%d ", Eq[combi[3]][j]);
                        printf("\n");
  } else printf("Disp: [94-92]: No existe combinacion en este rango!\n");

  if (costo[4] != 99999) {
                        printf("Disp:  [92-90]: %d\t%2.2f %2.2f\t", combi[4], disp[4]*100,
costo[4]);
                        for (j=0; j<5; j++) printf("%d ", CRedundantes[combi[4]][j]);
                        printf("\t");
                        for (j=0; j<5; j++) printf("%d ", Eq[combi[4]][j]);
                        printf("\n");
  } else printf("Disp: [92-90]: No existe combinacion en este rango!\n");

   if (costo[5] != 99999) {
                        printf("Disp:  [90-88]: %d\t%2.2f %2.2f\t", combi[5], disp[5]*100,
costo[5]);
                        for (j=0; j<5; j++) printf("%d ", CRedundantes[combi[5]][j]);
                        printf("\t");
                        for (j=0; j<5; j++) printf("%d ", Eq[combi[5]][j]);
                        printf("\n");
  } else printf("Disp: [90-88]: No existe combinacion en este rango!\n");

    if (costo[6] != 99999) {
                        printf("Disp:  [88-86]: %d\t%2.2f %2.2f\t", combi[6], disp[6]*100,
costo[6]);
                        for (j=0; j<5; j++) printf("%d ", CRedundantes[combi[6]][j]);
                        printf("\t");
                        for (j=0; j<5; j++) printf("%d ", Eq[combi[6]][j]);
                        printf("\n");
  } else printf("Disp: [88-86]: No existe combinacion en este rango!\n");
```

```
                        if (costo[7] != 99999) {
                        printf("Disp:  [86-84]: %d\t%2.2f %2.2f\t", combi[7], disp[7]*100,
costo[7]);
                        for (j=0; j<5; j++) printf("%d ", CRedundantes[combi[7]][j]);
                        printf("\t");
                        for (j=0; j<5; j++) printf("%d ", Eq[combi[7]][j]);
                        printf("\n");
  } else printf("Disp: [86-84]: No existe combinacion en este rango!\n");

                         if (costo[8] != 99999) {
                        printf("Disp:  [84-82]: %d\t%2.2f %2.2f\t", combi[8], disp[8]*100,
costo[8]);
                        for (j=0; j<5; j++) printf("%d ", CRedundantes[combi[8]][j]);
                        printf("\t");
                        for (j=0; j<5; j++) printf("%d ", Eq[combi[8]][j]);
                        printf("\n");
  } else printf("Disp: [84-82]: No existe combinacion en este rango!\n");

                          if (costo[9] != 99999) {
                        printf("Disp:  [82-80]: %d\t%2.2f %2.2f\t", combi[9], disp[9]*100,
costo[9]);
                        for (j=0; j<5; j++) printf("%d ", CRedundantes[combi[9]][j]);
                        printf("\t");
                        for (j=0; j<5; j++) printf("%d ", Eq[combi[9]][j]);
                        printf("\n");
  } else printf("Disp: [82-80]: No existe combinacion en este rango!\n");

}

/* Imprime los resultados a pantalla y de ser posible a un archivo de texto */
void ImprimirResultadosAM() {
  int i, j;


printf("============================================================\n");
  printf("Indice\tCRedun\t\tEqi\t\tRAs\t\tAs\tRCT\tCT\n");
  printf("============================================================\n");
  for (i=0; i<NUMCOMBINACIONES; i++) {
                        printf("%d\t", i);
    for (j=0; j<5; j++) {
                          printf("%d ", CRedundantes[i][j]);
                        }
                        printf("\t");
                        for (j=0; j<5; j++) {
                          printf("%d ", Eq[i][j]);
                        }
                        printf("\t%d", posDisponibilidad(i));
                        printf("\t");
```

```
                    printf("%2.2f%", Disponibilidad[i][5]*100);
                    printf("\t%d", posCosto(i));
                    printf("\t");
                    printf("%2.2f\n", Costos[i][4]);
                    printf("-----------------------------------------------------------------\n");
  }
  imprimeresultadoarchivo("pobAM.txt");
}

/*****************************************************************************/
/********** FUNCIONES ESPECIALIZADAS PARA EL SISTEMA *********/
/*****************************************************************************/

/* Lectura de los datos desde el archivo */
void LeerDatos(char NomArchivo[15]) {
  int i, j;
  FILE *fp;

  /* abrir el archivo */
  if ((fp = fopen(NomArchivo, "r")) == NULL) {
    printf("Error en la apertura del archivo de datos: %s!\n", NomArchivo);
                    exit(0);
  }

  printf("MTTF\tMTTR\tPeso\tVol\tCD\tCM\tCostEqMant\n");
  printf("=============================================================\n");
  /* leer datos desde el archivo */
  for (i=0; i<5; i++) {
    for (j=0; j<7; j++) {
                      fscanf(fp, "%d", &Datos[i][j]);
                      printf("%d\t", Datos[i][j]);
    }
                    printf("\n");
  }
  printf("=============================================================\n\n");

  /* cerrar el archivo */
  fclose(fp);
}

/* Se calculan los componentes redundantes y equipos de mantenimiento */
void GenerarNumerosAleatorios(int combinacion) {
  int i, valor, cotadeEq, suma;

  /*El minimo valor permitido es de 1, y 1*5 posiciones nos limita a 15 valores en Eq */
  cotadeEq = 16;

  for (i=0; i<5; i++) Eq[combinacion][i] = 1;
```

```
  for (i=0; i<5; i++) {
    CRedundantes[combinacion][i] = aleatorio(10);
                        valor = aleatorio(CRedundantes[combinacion][i]); /* Se obtiene un valor
menor o igual que el CR */
                         if ((valor > 1) && (valor <= cotadeEq)) {
                         Eq[combinacion][i] = valor;
                         cotadeEq = cotadeEq - valor + 1;
                        }
  }

  suma=0;
  for (i=0; i<5; i++)
                        suma += Eq[combinacion][i];

  Eq[combinacion][5] = suma;
}

/* Se calcula el Factor F para una determinada combinacion */
void CalcularFactorF(int combinacion) {
  int j;
  double ValorEncontrado, ParteEntera, ParteFracc;

  for (j=0; j<5; j++) { /* se recorren todos los pares de cada combinacion */
    ValorEncontrado = CRedundantes[combinacion][j] / Eq[combinacion][j];
    ParteFracc = modf(ValorEncontrado, &ParteEntera);

    if (ParteFracc == 0)
                         FactorF[combinacion][j] = (int) ValorEncontrado;
    else
                         FactorF[combinacion][j] = (int) (ParteEntera + 1);
  }

}

/* Se calcula el tiempo de reparacion para una determinada combinacion */
void CalcularMTTR1(int combinacion) {
  int j;

  for (j=0; j<5; j++) { /* se recorren todos los pares de cada combinacion */
    MTTR1[combinacion][j] = Datos[j][1] / CRedundantes[combinacion][j];
  }
}

/* Se calcula la confiabilidad para una determinada combinacion */
void CalcularConfiabilidad(int combinacion) {
  int j;

  for (j=0; j<5; j++) { /* se recorren todos los pares de cada combinacion */
```

```
   Confiabilidad[combinacion][j] = Datos[j][0] /
(FactorF[combinacion][j]*MTTR1[combinacion][j]);
                    Confiabilidad[combinacion][j] = (double)Confiabilidad[combinacion][j] /
CRedundantes[combinacion][j];
  }
}

/* Se calcula la disponibilidad para una determinada combinacion */
void CalcularDisponibilidad(int combinacion) {
  int j;
  double base, calculo, potencia;
  int exponente;


  Disponibilidad[combinacion][5] = 1;
  for (j=0; j<5; j++) { /* se recorren todos los pares de cada combinacion */
    calculo = 1.0 + ((1.0/FactorF[combinacion][j])*(double)Confiabilidad[combinacion][j]);
                    base = 1.0 / (double)calculo;
                    exponente = CRedundantes[combinacion][j];
                    potencia = elevar(base, exponente);
    Disponibilidad[combinacion][j] = 1.0 - potencia;
                    Disponibilidad[combinacion][5] = Disponibilidad[combinacion][5] *
Disponibilidad[combinacion][j];
  }

}

/* Se calcula la probabilidad de falla pra un determinado para de una combinacion */
double ProbabilidadFalla(int par) {
  return (double)(  1 - (double)exp( -((double)1/(double)Datos[par][0])*(double)T )  );
}

/* Calcula los costos de cada combinacion. 1-calculo correcto, 0-calculo incorrecto */
int CalcularCostos(int combinacion) {
  double CostoDiseno, Peso, Volumen, CostoMantenimiento1, CostoMantenimiento2,
CostoTotal;
  int j;

  CostoDiseno = 0;
  Peso = 0;
  Volumen = 0;
  CostoMantenimiento1 = 0;
  CostoMantenimiento2 = 0;
  CostoTotal = CostoDiseno,CostoMantenimiento1,CostoMantenimiento2;
  for (j=0; j<5; j++) { /* se recorren todos los pares de cada combinacion */
    CostoDiseno = CostoDiseno + (Datos[j][4]*CRedundantes[combinacion][j]);
    Peso = Peso + (Datos[j][2]*CRedundantes[combinacion][j]);
    Volumen = Volumen + (Datos[j][3]*CRedundantes[combinacion][j]);
    CostoMantenimiento1 = CostoMantenimiento1 + (Eq[combinacion][j]*Datos[j][6]);
```

```
    CostoMantenimiento2 = CostoMantenimiento2 +
(ProbabilidadFalla(j)*CRedundantes[combinacion][j]*Datos[j][5]);
    CostoTotal= CostoDiseno + CostoMantenimiento1 + CostoMantenimiento2;
  }
  Costos[combinacion][0] = CostoDiseno;
  Costos[combinacion][1] = Peso;
  Costos[combinacion][2] = Volumen;
  Costos[combinacion][3] = CostoMantenimiento1 + CostoMantenimiento2;
  Costos[combinacion][4] = CostoTotal;
  if ((Costos[combinacion][1] > 1000) || (Costos[combinacion][2] > 1000)) return(0);
  else return(1);
}

/* Se generan dos indices de ordenamiento, basados en el costo y la disponibilidad */
void GenerarIndiceOrdenamiento() {
  int i, j;

  for (i=0; i<NUMCOMBINACIONES; i++) {;
                        Cos[i][0] = i;
                        Cos[i][1] = Costos[i][4];
                        Dis[i][0] = i;
                        Dis[i][1] = Disponibilidad[i][5];
  }

  for (i=0; i<NUMCOMBINACIONES; i++) {
                        for (j=0; j<(NUMCOMBINACIONES-1); j++) {
                          /* Ranking para costo; de menor a mayor*/
                          if (Cos[j][1] > Cos[j+1][1]) {
                                        Swap(&Cos[j][0], &Cos[j+1][0]);
                                        Swap(&Cos[j][1], &Cos[j+1][1]);
                          }
                          /* Ranking para disponibilidad; de mayor a menor */
                          if (Dis[j][1] < Dis[j+1][1]) {
                                        Swap(&Dis[j][0], &Dis[j+1][0]);
                                        Swap(&Dis[j][1], &Dis[j+1][1]);
                          }
                        }
  }
}

/* Procedimiento que elige las cinco mejores combinaciones, basado en el costo */
void ElegirMasAptos() {
  GenerarIndiceOrdenamiento();
}

/* Realiza la cruza de dos combinaciones, arrojando dos hijos como resultado */
void Cruza() {
  int Padre1, Padre2;
  int componente1, componente2;
```

```
  int i, SumaTotal;

  /* Se escogen dos padres de manera aleatoria */
  Padre1 = aleatorio(5)-1;
  do {
    Padre2 = aleatorio(5)-1;
  } while (Padre2 == Padre1);

  /* Se copia la informacion de los padres en los hijos */
  for (i=0; i<5; i++) {
    CRedundantes[NUMCOMBINACIONES][i] = CRedundantes[(int)Dis[Padre1][0]][i];
                        CRedundantes[NUMCOMBINACIONES+1][i] =
CRedundantes[(int)Dis[Padre2][0]][i];
                        Eq[NUMCOMBINACIONES][i] = Eq[(int)Dis[Padre1][0]][i];
                        Eq[NUMCOMBINACIONES+1][i] = Eq[(int)Dis[Padre2][0]][i];
  }

  /* Se realiza el procedimiento de cruza */
  /* Se almacenan los valores hijo en las ultimas dos posiciones de la matriz de poblacion
*/
  componente1 = aleatorio(5)-1;
  componente2 = aleatorio(5)-1;
  /* Se realiza un intercambio del componente redundante */
  SwapInt(&CRedundantes[NUMCOMBINACIONES][componente1],
&CRedundantes[NUMCOMBINACIONES+1][componente2]);
  /* Se realiza un intercambio del correspondiente Eqi */
  SwapInt(&Eq[NUMCOMBINACIONES][componente1],
&Eq[NUMCOMBINACIONES+1][componente2]);

  /* Se realizan los correspondientes calculos para cada hijo generado */
  CalcularFactorF(NUMCOMBINACIONES);
  CalcularMTTR1(NUMCOMBINACIONES);
  CalcularConfiabilidad(NUMCOMBINACIONES);
  CalcularDisponibilidad(NUMCOMBINACIONES);
  if (!CalcularCostos(NUMCOMBINACIONES)) Disponibilidad[NUMCOMBINACIONES][5]
= 0;

  CalcularFactorF(NUMCOMBINACIONES+1);
  CalcularMTTR1(NUMCOMBINACIONES+1);
  CalcularConfiabilidad(NUMCOMBINACIONES+1);
  CalcularDisponibilidad(NUMCOMBINACIONES+1);
  if (!CalcularCostos(NUMCOMBINACIONES+1))
Disponibilidad[NUMCOMBINACIONES+1][5] = 0;
  /* Se verifica que la suma de los equipos sea menor o igual a 20 */
  /* Primer hijo*/
  SumaTotal = 0;
  for (i=0; i<5; i++) SumaTotal += Eq[NUMCOMBINACIONES][i];
  if (SumaTotal > 20) Disponibilidad[NUMCOMBINACIONES][5] = 0;
  /* Segundo hijo*/
```

```
  SumaTotal = 0;
  for (i=0; i<5; i++) SumaTotal += Eq[NUMCOMBINACIONES+1][i];
  if (SumaTotal > 20) Disponibilidad[NUMCOMBINACIONES+1][5] = 0;
}

/* Inserta un hijo en la poblacion */
void InsertaEnLaPoblacion(hijo, posicion) {
  int i;

  /* copia los valores de CRedundantes y Eqi */
  for (i=0; i<5; i++) {
                        CRedundantes[posicion][i] = CRedundantes[hijo][i];
                        Eq[posicion][i] = Eq[hijo][i];
  }

  /* Recalcular valores para los nuevos hijos */
  CalcularFactorF(posicion);
  CalcularMTTR1(posicion);
  CalcularConfiabilidad(posicion);
  CalcularDisponibilidad(posicion);
  if (!CalcularCostos(posicion)) Disponibilidad[posicion][5] = 0;
}

/* Se evalua el costo de los hijos generados y si se necesita, se insertan en la poblacion
*/
int EvaluarAptitud() {
  int PeorHijo, MejorHijo;
  int PeorCombinacion, SegundaPeorCombinacion;

  PeorCombinacion = (int)Dis[NUMCOMBINACIONES-1][0];
  SegundaPeorCombinacion = (int)Dis[NUMCOMBINACIONES-2][0];

   /* En caso contrario, si alguno mejora la poblacion, se regresa un cero */
  if (Disponibilidad[NUMCOMBINACIONES][5] <
Disponibilidad[NUMCOMBINACIONES+1][5]) {
                        PeorHijo = NUMCOMBINACIONES;
                        MejorHijo = NUMCOMBINACIONES+1;
  } else {
    PeorHijo = NUMCOMBINACIONES+1;
                        MejorHijo = NUMCOMBINACIONES;
  }

                        /* Se insertan los dos hijos */
                        InsertaEnLaPoblacion(PeorHijo, PeorCombinacion);
                        InsertaEnLaPoblacion(MejorHijo, SegundaPeorCombinacion);

   return(0);
}
```

```
void Mutar80porciento(int combinacion) {
  int i, valor, cotadeEq, suma, posicionaleatoria;

  /* Calculamos el valor de la posicion a mantener, de manera aleatoria */
  posicionaleatoria = aleatorio(5) - 1;

  for (i=0; i<5; i++) {
                        if (i != posicionaleatoria)               Eq[combinacion][i] = 1;
  }
  /*El minimo valor permitido es de 1, y 1*5 posiciones nos limita a 15 valores en Eq
CHECK*/
  cotadeEq = 16 - Eq[combinacion][posicionaleatoria];

  for (i=0; i<5; i++) {
                        if (i != posicionaleatoria) {
      CRedundantes[combinacion][i] = aleatorio(10);
                          valor = aleatorio(CRedundantes[combinacion][i]); /* Se obtiene un
valor menor o igual que el CR */
                           if ((valor > 1) && (valor <= cotadeEq)) {
                             Eq[combinacion][i] = valor;
                             cotadeEq = cotadeEq - valor + 1;
                           }
                        }
  }

  suma=0;
  for (i=0; i<5; i++)
                        suma += Eq[combinacion][i];

  Eq[combinacion][5] = suma;
}

/* Se aplica mutacion */
void Mutacion() {
  int i;

  /* Se aplica mutacion al 80% de la poblacion */
  for (i=5; i<NUMCOMBINACIONES; i++) {
    do {
                          Mutar80porciento(i);
      CalcularFactorF(i);
      CalcularMTTR1(i);
      CalcularConfiabilidad(i);
      CalcularDisponibilidad(i);
    } while (!CalcularCostos(i));
  }
}

/* Procedimiento basado en algoritmos geneticos */
```

```
void AplicarGeneticos() {
  int NumGeneraciones, GeneracionesSinCambio=0, mutacion=0;
  int i, primeravez=1;

  printf("Dar el numero de generaciones: ");
  scanf("%d", &NumGeneraciones);

  for (i=0; i<NumGeneraciones; i++) {
    ElegirMasAptos();
                        Cruza();
                        GeneracionesSinCambio += EvaluarAptitud();
                        /*if (GeneracionesSinCambio > (int)(NumGeneraciones*0.10)) {*/
                        if (!fmod(i, (int)(NumGeneraciones*0.10))) {
                          printf("********** En la  generacion %d  se aplica MUTACION %d
**********\n", i, mutacion++);
                          if (primeravez) { ImprimirResultadosAM(); primeravez=0;}
                          Mutacion();
                        }
  }
}

/* Se genera la poblacion inicial */
void ObtenPoblacion() {
  int i;

  for (i=0; i<NUMCOMBINACIONES; i++) {
                        do {
                          GenerarNumerosAleatorios(i);
      CalcularFactorF(i);
      CalcularMTTR1(i);
      CalcularConfiabilidad(i);
      CalcularDisponibilidad(i);
    } while (!CalcularCostos(i));
  }
}

/*****************************************************************************/
/*************** PROCEDIMIENTO PRINCIPAL ************************/
/*****************************************************************************/
void main(int argc, char *argv[]) {

  Inicializa();
  if (argc != 2) {printf("Debe dar el nombre del archivo de datos!\n"); exit(0);}
  LeerDatos(argv[1]);
  ObtenPoblacion();
  ImprimePoblacion();
  AplicarGeneticos();
  ImprimirResultados();
}
```