F.1 Source code of the ID3 algorithm

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>

//#define DEBUG

extern unsigned _stklen = 50000U;

/************************************************************************/
int id3(int file_code)
{
#define M1 15   // no of rows in learn.dat > no of items clustered
#define M2 150  // no of atts for ID3 > atts in attrib.dat                              //it was 150
before
#define M3 5    // length of string describing attribute                                //it was 4
before
#define M4 141  // no of attributes in attrib.dat + 1 (for EOS)       //it was 101 before

int data_load(char string[M1][M2][M3], int *, int *, char title[M2][M3]);
int check_all_positive(char string[M1][M2][M3], int, int);
int check_all_negative(char string[M1][M2][M3], int, int);
int get_diff_att_types(char valid[M1], char string[M1][M2][M3],
                                             char att_names[M1][M3], int, int);
int create_tree(char rule[M4], char avail_att[M2], FILE *ofp,
                                             FILE *nfp, FILE *pfp, char string[M1][M2][M3],
                                             char valid[M1], int, int, char title[M2][M3], int);
int not_all_same(char valid[M1], char string[M1][M2][M3], int attributes);

int attributes = 0, rows = 0, tab_cnt = 0;
char string[M1][M2][M3];
char title[M2][M3];
char valid[M1];
char avail_att[M2];
char rule[M4];
FILE *ofp, *pfp, *nfp;

if (data_load(string,&attributes,&rows,title) == 999)
                    {
        //printf("load\n");
                    return 0;
                    }

//printf("%d\n", attributes);
//printf("%d\n", rows);

if (file_code == 1)
          {
         if ((ofp = fopen("d_tree.dat","w")) == NULL)
                    {
                    printf("File error : Cannot create output file TREE.DAT\n");
                    return 0;
                    }
          if ((pfp = fopen("d_pos.dat","w")) == NULL)
                    {
                    printf("File error : Cannot create output file POSITIVE.DAT\n");
                    return 0;
                    }
          if ((nfp = fopen("d_neg.dat","w")) == NULL)
                    {
                    printf("File error : Cannot create output file NEGATIVE.DAT\n");
                    return 0;
                    }
         }
else
          {
         if ((ofp = fopen("d_tree.dat","a")) == NULL)
                    {
                    printf("File error : Cannot create output file TREE.DAT\n");
                    return 0;
                    }
          if ((pfp = fopen("d_pos.dat","a")) == NULL)
                    {
                    printf("File error : Cannot create output file POSITIVE.DAT\n");
                    return 0;
                    }
          if ((nfp = fopen("d_neg.dat","a")) == NULL)
                    {
                    printf("File error : Cannot create output file NEGATIVE.DAT\n");
                    return 0;
                    }
         }


fprintf(pfp, "rule\n");
fprintf(nfp, "rule\n");
fprintf(ofp, "\n");

if (check_all_positive(string,attributes,rows))
                    {
                    fprintf(ofp,"HALT:all_positive\n");
                    fprintf(pfp,"HALT:all_positive\n");
                    fprintf(nfp,"HALT:all_positive\n");
                    fprintf(pfp, "rule_end\n");
                    fprintf(nfp, "rule_end\n");
                    fclose(ofp);
                    fclose(nfp);
                    fclose(pfp);
                    return 1;
                    }

if (check_all_negative(string,attributes,rows))
                    {
                    fprintf(ofp,"HALT:all_negative\n");
                    fprintf(pfp,"HALT:all_negative\n");
                    fprintf(nfp,"HALT:all_negative\n");
                    fprintf(pfp, "rule_end\n");
                    fprintf(nfp, "rule_end\n");
                    fclose(ofp);
                    fclose(nfp);
                    fclose(pfp);
                    return 1;
                    }

memset (valid, 42, rows);    // set to '*'
memset (avail_att, 42, M2);  // set to '*'
memset (rule, 45, M4);       // set to '-'

if (create_tree(rule, avail_att, ofp, nfp, pfp, string, valid, rows, attributes, title, tab_cnt) == 999)
                    {
                    return 0;
                    }

fprintf(pfp, "rule_end\n");
fprintf(nfp, "rule_end\n");

fclose(ofp);
fclose(pfp);
fclose(nfp);

return 1;
}

/***************************************************************************/
int create_tree(char rule[M4], char avail_att[M2], FILE *ofp,
                                             FILE *nfp, FILE *pfp, char string[M1][M2][M3],
                                             char valid[M1], int rows, int attributes, char
title[M2][M3],
                                             int tab_cnt)
{
int get_diff_att_types(char valid[M1], char string[M1][M2][M3],
                                             char att_names[M1][M3], int, int);
int not_all_same(char valid[M1], char string[M1][M2][M3], int attributes);
int find_att(char avail_att[M2], char string[M1][M2][M3], char valid[M1],
                                              int attributes, int rows);

char att_names[M1][M3] = {" "};
char valid_2[M1];
char avail_att_2[M2];
char rule_2[M4];
int  j, l, i, ret, tot_diff_atts, att_no;

for (i=0;i<tab_cnt+tab_cnt;++i)
                    {
                    fprintf(ofp,"\t");
                    }

tab_cnt++;

if ((att_no = find_att(avail_att, string, valid, attributes, rows)) == 999)
                    {
        //printf("attno\n");
                    return 999;
                    }

rule[M4-1] = '\0';                 //make string
avail_att[M2-1] = '\0';            //make string
strcpy(avail_att_2, avail_att);
avail_att_2[att_no] = ' ';

fprintf(ofp, "[%s]\n", title[att_no]);

tot_diff_atts = get_diff_att_types(valid, string, att_names, att_no, rows);

for (j=0;j<tot_diff_atts;++j)
                    {
```

```
                    valid[M1-1] = '\0';
                    strcpy(valid_2,valid);

                    for (l=0;l<rows;++l)
                                             {
                                             if (strcmp(att_names[j],string[l][att_no]) != 0)
                                                                      {
                                                                      valid_2[l] = ' ';
                                                                      }
                                             }

                    if ((ret = not_all_same(valid_2,string,attributes)) == 1)
                                             {
                                             for (i=0;i<tab_cnt+tab_cnt-1;++i)
                                                                      {
                                                                      fprintf(ofp,"\t");
                                                                      }

                                             fprintf(ofp," %s\n",att_names[j]);
                                             rule[att_no-1] = att_names[j][0];
                                             strcpy(rule_2, rule);

                                             if (create_tree(rule_2, avail_att_2, ofp, nfp, pfp,
                                                                      string, valid_2, rows,
attributes,title,tab_cnt) == 999)
                                                                      {
                                                                      return 999;
                                                                      }
                                             }
                    else
                                             {
                                             for (i=0;i<tab_cnt+tab_cnt-1;++i)
                                                                      {
                                                                      fprintf(ofp,"\t");
                                                                      }
                                             if (ret == 2)
                                                                      {
                                                                      fprintf(ofp," %s\t -
YES\n",att_names[j]);
                                                                      rule[att_no-1] = att_names[j][0];
                                                                      fprintf(pfp,"%s\n",rule);
                                                                      }
                                             else
                                                                      {
                                                                      fprintf(ofp," %s\t -
NO\n",att_names[j]);
                                                                      rule[att_no-1] = att_names[j][0];
                                                                      fprintf(nfp,"%s\n",rule);
                                                                      }
                                             }
                    }
return 1;
}

/*************************************************************************/
int find_att(char avail_att[M2], char string[M1][M2][M3], char valid[M1],
                                             int attributes, int rows)
{

int get_diff_att_types(char valid[M1], char string[M1][M2][M3],
                                             char att_names[M1][M3], int, int);
void disaster(int);

int i, j, l, y_tot = 0, n_tot = 0, y_tot_2, n_tot_2;
int tot_diff_atts;
int att_no = 0;
double max_inf_gain = -1.0;
double entropy, entropy_2, r_entropy_tot;
double att_entropy[M2];
char att_names[M1][M3] = {" "};
char valid_2[M1];

// CHOSE ONE OF THE FOLLOWING

// THIS IS THE MAX INFO GAIN

for (i=0;i<M2;++i)
                    {
                    att_entropy[i] = -2.0;
                    }

// THIS IS THE MIN INFO GAIN

//for (i=0;i<M2;++i)
//                  {
//                  att_entropy[i] = 2.0;
//                  }

// CHOSE ONE OF THE ABOVE

for (i=1;i<=M1;i++)
```

```
                    {
                    if (valid[i] == '*')
                                              {
                                              if (strcmp(string[i][attributes],"yes") == 0)
                                                                      ++y_tot;
                                              if (strcmp(string[i][attributes],"no") == 0)
                                                                      ++n_tot;
                                              }
                    }

if (y_tot == 0 || n_tot == 0)
                    entropy = 0.0;
else
                    {
                    entropy =  0.0 - ((y_tot/(double)(y_tot+n_tot))

         *log((y_tot/(double)(y_tot+n_tot))))
                                                                      - ((n_tot/(double)(y_tot+n_tot))*

         log((n_tot/(double)(y_tot+n_tot))));
                    }

for (i=1;i<attributes;++i)
                    {
                    if (avail_att[i] == '*')
                                              {
                                              r_entropy_tot = 0.0;
                                              tot_diff_atts = get_diff_att_types(valid, string,

                                                                att_names, i, rows);

                                              for (j=0;j<tot_diff_atts;++j)
                                                                      {

                                                                      memset (valid_2, 32, M1);

                                                                      for (l=1;l<=rows;++l)
                                                                                          {
                                                                                          if
((strcmp(att_names[j],string[l][i]) == 0)

                   && (valid[l] == '*'))

                   valid_2[l] = '*';

                                                                                          }

                                                                      y_tot_2 = 0;
                                                                      n_tot_2 = 0;

                                                                      for (l=1;l<=M1;l++)
                                                                                          {
                                                                                          if (valid_2[l]
== '*')

                   {

                   if (strcmp(string[l][attributes],"yes") == 0)

                                             ++y_tot_2;

                   if (strcmp(string[l][attributes],"no") == 0)

                                             ++n_tot_2;

                   }
                                                                                          }

                                                                      if (n_tot_2 == 0 || y_tot_2 == 0)
                                                                                          entropy_2 =
0.0;
                                                                      else
                                                                                          {
                                                                                          entropy_2 =
0.0 - ((y_tot_2/(double)(y_tot_2+n_tot_2))

                                              *log((y_tot_2/(double)(y_tot_2+n_tot_2))))

                                              - ((n_tot_2/(double)(y_tot_2+n_tot_2))

                                              *log((n_tot_2/(double)(y_tot_2+n_tot_2))));
                                                                                          }

                                                                      r_entropy_tot = r_entropy_tot +
(entropy_2

                   * ((n_tot_2+y_tot_2)/(double)(n_tot+y_tot)));
                                                                      }

                                              att_entropy[i] = entropy - r_entropy_tot;

                                              }
                    }

// CHOSE ONE OF THE FOLLOWING

// THIS IS THE MAX INFO GAIN

for (l=0;l<M2;++l)
                    {
                    if (att_entropy[l] >= max_inf_gain)
                                              {
                                              max_inf_gain = att_entropy[l];
                                              att_no = l;
                                              }
                    }

if (max_inf_gain == 0.0)
                    {
                    disaster(1);
                    return 999;
                    }

// THIS IS THE MIN INFO GAIN

//max_inf_gain = 1.99;

//for (l=0;l<M2;++l)
//                  {
//                  if (att_entropy[l] <= max_inf_gain)
//                                            {
//                                            max_inf_gain = att_entropy[l];
//                                            att_no = l;
//                                            }
//                  }

//if (max_inf_gain == 1.99)
//                  {
//                  disaster(1);
//                  return 999;
//                  }

// CHOOSE ONE OF THE ABOVE

return att_no;

}

/****************************************************************************/
int not_all_same(char valid[M1], char string[M1][M2][M3], int attributes)
{
int i, y_tot = 0, n_tot = 0;

for (i=0;i<M1;i++)
                    {
                    if (valid[i] == '*')
                                              {
                                              if (strcmp(string[i][attributes],"yes") == 0)
                                                                        ++y_tot;
                                              if (strcmp(string[i][attributes],"no") == 0)
                                                                        ++n_tot;
                                              }
                    }

if (n_tot == 0)
                    return 2;    /* all yes */
else if (y_tot == 0)
                    return 3;    /* all no  */
else
                    return 1;
}

/****************************************************************************/
int get_diff_att_types(char valid[M1], char string[M1][M2][M3],
                                                                                        char
att_names[M1][M3], int att, int max_row)
{
int j,l,k;
char att_temp[M1][M3];

for(j=0;j<max_row;j++)
                    {
                    strcpy(att_names[j],string[j][att]);
                    }

for(l=0;l<j;++l)
                    {
                    if (valid[l] != '*')
                                               memset(att_names[l], 42, M3-1);
                    }

for(j=0;j<max_row;j++)
                    {
```

```
                    l=1;
                    for(l=l+j;l<max_row;l++)
                                          {
                                          if (strcmp(att_names[j],att_names[l]) == 0)
                                                                    {
                                                                    memset(att_names[l], 42, M3-1);
                                                                    }
                                          }
                    }

for(l=0,k=0;l<j;l++)
                    {
                    if (att_names[l][0] != '*')
                                          {
                                          strcpy(att_temp[k],att_names[l]);
                                          k++;
                                          }
                    }

for(l=0;l<j;++l)
                    {
                    memset(att_names[l], 42, M3-1);
                    }

for(l=0;l<k;++l)
                    {
                    strcpy(att_names[l],att_temp[l]);
                    }

for(l=0,k=0;l<j;l++)
                    {
                    if (att_names[l][0] != '*')
                                          ++k;
                    }

return k;

}

/*************************************************************************/
int check_all_positive(char string[M1][M2][M3], int attributes, int rows)
{
int i;

for(i=0;i<rows;++i)
                    {
                    if (strcmp(string[i][attributes],"no") == 0)
                                          {
                                          return 0;
                                          }
                    }
return 1;
}

/*************************************************************************/
int check_all_negative(char string[M1][M2][M3], int attributes, int rows)
{
int i;

for(i=0;i<rows;++i)
                    {
                    if (strcmp(string[i][attributes],"yes") == 0)
                                          {
                                          return 0;
                                          }
                    }
return 1;
}

/*************************************************************************/
int data_load(char string[M1][M2][M3], int* a, int* b, char title[M2][M3])
{
char linebuff[20];
int k=0;

FILE *ifp;

if ((ifp = fopen("d_learn.dat","r")) == NULL)
                    {
                    printf("File error : Cannot open input file LEARN.DAT\n");
                    return 999;
                    }

do {
                    fscanf(ifp,"%s",title[k]);
         }while(title[k++][0] != '*');

do {
                    (*a)=0;

                    do    {
```

```
                                    fscanf(ifp,"%s",linebuff);
                                    strcpy(string[(*b)][(*a)],linebuff);
                                    (*a)++;
                                    }while(linebuff[0] != '*');

                    (*b)++;

         }while(linebuff[1] !='*');

*a = *a - 2;

fclose(ifp);

return 1;
}

/*****************************************************************************/
void disaster(int i)
{

switch(i)
                    {
                    case 1: printf("** ID3 failure **\n");
                                                              //system("cls");
                                             //printf("\nA serious error has occured.\n\n");
                                             //printf("All output files may be corrupt.\n\n");
                                             //printf("Possible inconistancies or contradictory\n");
                                             //printf("input cases may be the cause.\n\n");
                                             //printf("\n\nPress any key");
                                             //getche();
                                             break;
                    }
}

/*****************************************************************************/
```

F.2 Source code of the SG-1 algorithm

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>

//#define DEBUG

extern unsigned _stklen = 50000U;

/*************************************************************************/
int id3(int file_code)
{
#define M1 15   // no of rows in learn.dat > no of items clustered
#define M2 150  // no of atts for ID3 > atts in attrib.dat                              //it was 150
before
#define M3 5    // length of string describing attribute                                //it was 4
before
#define M4 141  // no of attributes in attrib.dat + 1 (for EOS)         //it was 101 before

int data_load(char string[M1][M2][M3], int *, int *, char title[M2][M3]);
int check_all_positive(char string[M1][M2][M3], int, int);
int check_all_negative(char string[M1][M2][M3], int, int);
int get_diff_att_types(char valid[M1], char string[M1][M2][M3],
                                         char att_names[M1][M3], int, int);
int create_tree(char rule[M4], char avail_att[M2], FILE *ofp,
                                         FILE *nfp, FILE *pfp, char string[M1][M2][M3],
                                         char valid[M1], int, int, char title[M2][M3], int);
int not_all_same(char valid[M1], char string[M1][M2][M3], int attributes);

int attributes = 0, rows = 0, tab_cnt = -1;
char string[M1][M2][M3];
char title[M2][M3];
char valid[M1];
char avail_att[M2];
char rule[M4];
FILE *ofp, *pfp, *nfp;

if (data_load(string,&attributes,&rows,title) == 999)
                  {
                  //printf("load\n");
                  return 0;
                  }

//printf("%d\n", attributes);
//printf("%d\n", rows);

if (file_code == 1)
         {
        if ((ofp = fopen("d_tree.dat","w")) == NULL)
                  {
                  printf("File error : Cannot create output file TREE.DAT\n");
                  return 0;
                  }
         if ((pfp = fopen("d_pos.dat","w")) == NULL)
                  {
                  printf("File error : Cannot create output file POSITIVE.DAT\n");
                  return 0;
                  }
         if ((nfp = fopen("d_neg.dat","w")) == NULL)
                  {
                  printf("File error : Cannot create output file NEGATIVE.DAT\n");
                  return 0;
                  }
         }
else
         {
         if ((ofp = fopen("d_tree.dat","a")) == NULL)
                  {
                  printf("File error : Cannot create output file TREE.DAT\n");
                  return 0;
                  }
         if ((pfp = fopen("d_pos.dat","a")) == NULL)
                  {
                  printf("File error : Cannot create output file POSITIVE.DAT\n");
                  return 0;
                  }
         if ((nfp = fopen("d_neg.dat","a")) == NULL)
                  {
                  printf("File error : Cannot create output file NEGATIVE.DAT\n");
                  return 0;
                  }
        }
```

```
fprintf(pfp, "rule\n");
fprintf(nfp, "rule\n");
fprintf(ofp, "\n");

if (check_all_positive(string,attributes,rows))
                  {
                  fprintf(ofp,"HALT:all_positive\n");
                  fprintf(pfp,"HALT:all_positive\n");
                  fprintf(nfp,"HALT:all_positive\n");
                  fprintf(pfp, "rule_end\n");
                  fprintf(nfp, "rule_end\n");
                  fclose(ofp);
                  fclose(nfp);
                  fclose(pfp);
                  return 1;
                  }

if (check_all_negative(string,attributes,rows))
                  {
                  fprintf(ofp,"HALT:all_negative\n");
                  fprintf(pfp,"HALT:all_negative\n");
                  fprintf(nfp,"HALT:all_negative\n");
                  fprintf(pfp, "rule_end\n");
                  fprintf(nfp, "rule_end\n");
                  fclose(ofp);
                  fclose(nfp);
                  fclose(pfp);
                  return 1;
                  }

memset (valid, 42, rows);    // set to '*'
memset (avail_att, 42, M2);  // set to '*'
memset (rule, 45, M4);       // set to '-'

if (create_tree(rule, avail_att, ofp, nfp, pfp, string, valid, rows, attributes, title, tab_cnt) == 999)
                  {
                  return 0;
                  }

fprintf(pfp, "rule_end\n");
fprintf(nfp, "rule_end\n");

fclose(ofp);
fclose(pfp);
fclose(nfp);

return 1;
}

/***************************************************************************/
int create_tree(char rule[M4],
                                         char avail_att[M2],
                                         FILE *ofp,
                                         FILE *nfp,
                                         FILE *pfp,
                                         char string[M1][M2][M3],
                                         char valid[M1],
                                         int rows, int attributes,
                                         char title[M2][M3],
                                         int tab_cnt)
        {
        int get_diff_att_types(char valid[M1],
                                                              char string[M1][M2][M3],
                                                              char att_names[M1][M3],
                                                              int,
                                                              int);

        int not_all_same(char valid[M1],
                                              char string[M1][M2][M3],
                                              int attributes);

        int find_att(char avail_att[M2],
                                         char string[M1][M2][M3],
                                         char valid[M1],
                                         int attributes,
                                         int rows,
                                         int function_code,
                                         int which_best);

        char att_names[M1][M3] = {" "};
        char valid_2[M1];
        char avail_att_2[M2];
        char rule_2[M4];
        char rule_work[M4];
        int j;
        int l;
        int i;
        int ret;
        int tot_diff_atts;
        int att_no;
        int function_code;
        int equal_best;
        int which_best;
        int for_each_rule;

//      for (i=0;i<tab_cnt+tab_cnt;++i)
//              {
//              fprintf(ofp,"\t");
//              }

        tab_cnt++;

        which_best = 999;
        function_code = 1;

        if ((equal_best = find_att(avail_att, string, valid, attributes, rows, function_code,
which_best)) == 999)
                {
                return 999;
                }

//      printf("Equal best : %d \n", equal_best); getche();

        for (for_each_rule = 1; for_each_rule <= equal_best; for_each_rule++)
                {
//              printf("for each rule : %d \n", for_each_rule); getche();

                function_code = 3;
                which_best = for_each_rule;

                if ((att_no = find_att(avail_att, string, valid, attributes, rows, function_code,
which_best)) == 999)
                        {
                        return 999;
                        }

                rule[M4-1] = '\0';                  //make string
                avail_att[M2-1] = '\0';             //make string
                strcpy(rule_work, rule);
                strcpy(avail_att_2, avail_att);

                avail_att_2[att_no] = ' ';

                for (i=0;i<tab_cnt+tab_cnt;++i)
                        {
                        fprintf(ofp,"\t");
                        }

//              tab_cnt++;

                fprintf(ofp, "[%s]\n", title[att_no]);

                tot_diff_atts = get_diff_att_types(valid, string, att_names, att_no, rows);

                for (j=0;j<tot_diff_atts;++j)
                        {
                        valid[M1-1] = '\0';
                        strcpy(valid_2,valid);
                        for (l=0;l<rows;++l)
                                {
                                if (strcmp(att_names[j],string[l][att_no]) != 0)
                                        {
                                        valid_2[l] = ' ';
                                        }
                                }
                        if ((ret = not_all_same(valid_2,string,attributes)) == 1)
                                {
                                for (i=0;i<tab_cnt+tab_cnt+1;++i)
                                        {
                                        fprintf(ofp,"\t");
                                        }
                                fprintf(ofp," %s\n",att_names[j]);
                                rule_work[att_no-1] = att_names[j][0];
                                strcpy(rule_2, rule_work);
                                if (create_tree(rule_2, avail_att_2, ofp, nfp, pfp,
                                                string, valid_2, rows, attributes,title,tab_cnt) == 999)
                                        {
                                        return 999;
                                        }
                                }
                        else
                                {
                                for (i=0;i<tab_cnt+tab_cnt+1;++i)
                                        {
                                        fprintf(ofp,"\t");
                                        }
                                if (ret == 2)
                                        {
                                        fprintf(ofp," %s\t - YES\n",att_names[j]);
                                        rule_work[att_no-1] = att_names[j][0];
                                        fprintf(pfp,"%s\n",rule_work);
                                        }
                                else
                                        {
                                        fprintf(ofp," %s\t - NO\n",att_names[j]);
                                        rule_work[att_no-1] = att_names[j][0];
                                        fprintf(nfp,"%s\n",rule_work);
                                        }
                                }
                        }
                } // end for each of the best

        return 1;
        }

/***********************************************************************/
int find_att(char avail_att[M2],
                                 char string[M1][M2][M3],
                                 char valid[M1],
                                 int attributes,
                                 int rows,
                                 int function_code,
                                 int which_best)
        {

        int get_diff_att_types(char valid[M1],
                                                                  char string[M1][M2][M3],
                                                                  char att_names[M1][M3],
                                                                  int,
                                                                  int);
        void disaster(int);

        int i, j, l, y_tot = 0, n_tot = 0, y_tot_2, n_tot_2;
        int tot_diff_atts;
        int att_no = 0;
        double max_inf_gain = -1.0;
        double entropy, entropy_2, r_entropy_tot;
        double att_entropy[M2];
        char att_names[M1][M3] = {" "};
        char valid_2[M1];
        int equal_best;

// CHOSE ONE OF THE FOLLOWING

// THIS IS THE MAX INFO GAIN

        for (i=0;i<M2;++i)
                {
                att_entropy[i] = -2.0;
                }

// THIS IS THE MIN INFO GAIN

// for (i=0;i<M2;++i)
//              {
//              att_entropy[i] = 2.0;
//              }

// CHOSE ONE OF THE ABOVE

        for (i=1;i<=M1;i++)
                {
                if (valid[i] == '*')
                        {
                        if (strcmp(string[i][attributes],"yes") == 0)
                                {
                                ++y_tot;
                                }
                        if (strcmp(string[i][attributes],"no") == 0)
                                {
                                ++n_tot;
                                }
                        }
                }

        if (y_tot == 0 || n_tot == 0)
                {
                entropy = 0.0;
                }
        else
                {
                entropy =  0.0 - ((y_tot/(double)(y_tot+n_tot))
                                                                 * log((y_tot/(double)(y_tot+n_tot))))
                                                                 - ((n_tot/(double)(y_tot+n_tot))
                                                                 * log((n_tot/(double)(y_tot+n_tot))));
                }

        for (i=1;i<attributes;++i)
                {
                if (avail_att[i] == '*')
                        {
                        r_entropy_tot = 0.0;
                        tot_diff_atts = get_diff_att_types(valid, string, att_names, i, rows);
                        for (j=0;j<tot_diff_atts;++j)
                                {
                                memset (valid_2, 32, M1);
                                for (l=1;l<=rows;++l)
                                        {
                                        if ((strcmp(att_names[j],string[l][i]) == 0) && (valid[l] ==
'*'))
                                                {
                                                valid_2[l] = '*';
                                                }
                                        }
                                y_tot_2 = 0;
                                n_tot_2 = 0;
                                for (l=1;l<=M1;l++)
                                        {
                                        if (valid_2[l] == '*')
                                                {
                                                if (strcmp(string[l][attributes],"yes") == 0)
                                                        {
                                                        ++y_tot_2;
                                                        }
                                                if (strcmp(string[l][attributes],"no") == 0)
                                                        {
                                                        ++n_tot_2;
                                                        }
                                                }
                                        }
                                if (n_tot_2 == 0 || y_tot_2 == 0)
                                        {
                                        entropy_2 = 0.0;
                                        }
                                else
                                        {
                                        entropy_2 =  0.0 - ((y_tot_2/(double)(y_tot_2+n_tot_2))
                                                                                                    *
log((y_tot_2/(double)(y_tot_2+n_tot_2))))
                                                                                                    -
((n_tot_2/(double)(y_tot_2+n_tot_2))
                                                                                                    *
log((n_tot_2/(double)(y_tot_2+n_tot_2))));
                                        }
                                r_entropy_tot = r_entropy_tot + (entropy_2
                                                                                      *
((n_tot_2+y_tot_2)/(double)(n_tot+y_tot)));
                                }
                                att_entropy[i] = entropy - r_entropy_tot;
                        }
                }

// CHOSE ONE OF THE FOLLOWING

// THIS IS THE MAX INFO GAIN

        equal_best = 0;

        for (l=0;l<M2;++l)
                {
                if (att_entropy[l] >= max_inf_gain)
                        {
//                      printf("Att entropy : %f \n", att_entropy[l]); getche();
//                      if ((att_entropy[l] == max_inf_gain) && (max_inf_gain >= 0.0))
//                              {
//                              equal_best++;
//                              }
                        max_inf_gain = att_entropy[l];
                        att_no = l;
                        }
                }

        if (function_code == 1)
                {
                for (l=0;l<M2;++l)
                        {
                        if (att_entropy[l] >= max_inf_gain)
                                {
//                              printf("Att entropy : %f \n", att_entropy[l]); getche();
                                equal_best++;
                                }
                        }
                }

//      printf("Max info gain : %f \n", max_inf_gain); getche();

        if (function_code == 3)
                {
                equal_best = 0;
                for (l=0;l<M2;++l)
                        {
//                      printf("Att entropy : %f \n", att_entropy[l]); getche();
                        if (att_entropy[l] >= max_inf_gain)
```

```
                                {
                                att_no = l;
                                equal_best++;
                                }
                        if (which_best == equal_best)
                                {
                                break;
                                }
                        }
                }

//      if (equal_best > 0)
//              {
//              printf("Equal best : %d %f %d \n", equal_best, max_inf_gain, att_no); getche();
//              }

        if (max_inf_gain == 0.0)
                {
                disaster(1); return 999;
                }

// THIS IS THE MIN INFO GAIN

// max_inf_gain = 1.99;

// for (l=0;l<M2;++l)
//         {
//         if (att_entropy[l] <= max_inf_gain)
//            {
//               max_inf_gain = att_entropy[l];
//               att_no = l;
//               }
//         }

// if (max_inf_gain == 1.99)
//         {
//         disaster(1);
//         return 999;
//         }

// CHOOSE ONE OF THE ABOVE

        if (function_code == 0)
                {
                return att_no;
                }

        if (function_code == 1)
                {
                return equal_best;
                }

        if (function_code == 3)
                {
                return att_no;
                }

        }
/*************************************************************************/
int not_all_same(char valid[M1], char string[M1][M2][M3], int attributes)
{
int i, y_tot = 0, n_tot = 0;

for (i=0;i<M1;i++)
                  {
                  if (valid[i] == '*')
                                       {
                                       if (strcmp(string[i][attributes],"yes") == 0)
                                                       ++y_tot;
                                       if (strcmp(string[i][attributes],"no") == 0)
                                                       ++n_tot;
                                       }
                  }

if (n_tot == 0)
                  return 2;    /* all yes */
else if (y_tot == 0)
                  return 3;    /* all no  */
else
                  return 1;
}

/*************************************************************************/
int get_diff_att_types(char valid[M1], char string[M1][M2][M3],
                                                                                  char
att_names[M1][M3], int att, int max_row)
{
int j,l,k;
char att_temp[M1][M3];
```

```
for(j=0;j<max_row;j++)
                  {
                  strcpy(att_names[j],string[j][att]);
                  }

for(l=0;l<j;++l)
                  {
                  if (valid[l] != '*')
                                          memset(att_names[l], 42, M3-1);
                  }

for(j=0;j<max_row;j++)
                  {
                  l=1;
                  for(l=l+j;l<max_row;l++)
                                         {
                                         if (strcmp(att_names[j],att_names[l]) == 0)
                                                                {
                                                                memset(att_names[l], 42, M3-1);
                                                                }
                                         }
                  }

for(l=0,k=0;l<j;l++)
                  {
                  if (att_names[l][0] != '*')
                                         {
                                         strcpy(att_temp[k],att_names[l]);
                                         k++;
                                         }
                  }

for(l=0;l<j;++l)
                  {
                  memset(att_names[l], 42, M3-1);
                  }

for(l=0;l<k;++l)
                  {
                  strcpy(att_names[l],att_temp[l]);
                  }

for(l=0,k=0;l<j;l++)
                  {
                  if (att_names[l][0] != '*')
                                         ++k;
                  }

return k;

}

/*************************************************************************/
int check_all_positive(char string[M1][M2][M3], int attributes, int rows)
{
int i;

for(i=0;i<rows;++i)
                  {
                  if (strcmp(string[i][attributes],"no") == 0)
                                         {
                                         return 0;
                                         }
                  }
return 1;
}

/*************************************************************************/
int check_all_negative(char string[M1][M2][M3], int attributes, int rows)
{
int i;

for(i=0;i<rows;++i)
                  {
                  if (strcmp(string[i][attributes],"yes") == 0)
                                         {
                                         return 0;
                                         }
                  }
return 1;
}

/*************************************************************************/
int data_load(char string[M1][M2][M3], int* a, int* b, char title[M2][M3])
{
char linebuff[20];
int k=0;

FILE *ifp;

if ((ifp = fopen("d_learn.dat","r")) == NULL)
```

```
                  {
                  printf("File error : Cannot open input file LEARN.DAT\n");
                  return 999;
                  }

do {
                  fscanf(ifp,"%s",title[k]);
        }while(title[k++][0] != '*');

do {
                  (*a)=0;

                  do    {
                                fscanf(ifp,"%s",linebuff);
                                strcpy(string[(*b)][(*a)],linebuff);
                                (*a)++;
                                }while(linebuff[0] != '*');

                  (*b)++;

        }while(linebuff[1] !='*');

*a = *a - 2;

fclose(ifp);

return 1;
}

/****************************************************************************/
void disaster(int i)
{

switch(i)
                  {
                  case 1: printf("** ID3 failure **\n");
                                                  //system("cls");
                                   //printf("\nA serious error has occured.\n\n");
                                   //printf("All output files may be corrupt.\n\n");
                                   //printf("Possible inconistancies or contradictory\n");
                                   //printf("input cases may be the cause.\n\n");
                                   //printf("\n\nPress any key");
                                   //getche();
                                   break;
                  }
}

/****************************************************************************/
```

F.3 Source code of the conceptual clustering algorithm

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>

#include "MLTClust.h"

//#define DEBUG

#define LIMIT1 10                                   // how many records input ie pages in input1.dat
#define LIMIT2 (((LIMIT1 * LIMIT1) - LIMIT1) / 2)   // required array size

int clust()
{

// Function prototype definitions

int find_cluster_diameter(char *, cluster_record *, attribute_record_out *);
int nearly_central(int, char *, cluster_record *, attribute_record_out *);
int find_distance_1(attribute_data, attribute_data);
int sort_function_1(cluster_record *, cluster_record *);
int sort_function_2(attribute_record_out *, attribute_record_out *);
attribute_data find_subj_attribute_vector(char *, FILE *);
void new_name(char *, int *);

// Local variables

FILE *ifp, *ofp, *afp;
int i, j, k, max, biggest, new_clust = 1, lines;
int last_page_num, latest, actual;
int cur_weight, max_weight, heaviest;
char buffer[8], temp_a[8], temp_b[8], last_page[8];
attribute_record_out attribute_rec_out[LIMIT1];
cluster_record sort_array[LIMIT2];

// Open the input and output attribute record files.

if ((ifp = fopen("d_input1.dat","r")) == NULL)
        {
        printf("File error : Cannot open input file INPUT1.DAT\n");
        return 0;
        }

if ((ofp = fopen("d_output.dat","w")) == NULL)
        {
        printf("File error : Cannot create output file OUTPUT.DAT\n");
        return 0;
        }

if ((afp = fopen("d_attrib.dat","r")) == NULL)
        {
        printf("File error : Cannot open attribute file ATTRIB.DAT\n");
        return 0;
        }

// Read the subj identifiers into memory and access the attribute
// descriptions from the subj attribute database.

fseek(ifp, -(LIMIT1 * 9), 2);

for (i=0; i<LIMIT1; ++i)
        {
        fscanf(ifp, "%s", attribute_rec_out[i].attribute_record_in.subj_id);
        attribute_rec_out[i].attribute_record_in.subj_attributes =
        find_subj_attribute_vector(attribute_rec_out[i].attribute_record_in.subj_id, afp);
        strcpy(attribute_rec_out[i].assigned_cluster_name, "DUMMY");
        attribute_rec_out[i].weight = i+1;
        if (i==LIMIT1-1)
                {
                strcpy(last_page,attribute_rec_out[i].attribute_record_in.subj_id);
                //printf("Last page id is : %s ", last_page);
                last_page_num = atoi(last_page+5);
                //printf("Last page id is : %d ", last_page_num);
                }
        }

// Load the sort array with pairs of unordered input points. Add the
// distance between the pairs to the array.

k = 0;
for (i=0; i<LIMIT1; ++i)
        {
        for (j=i+1; j<LIMIT1; ++j)
                {
```

```
                sort_array[k].input_case_a = i;
                sort_array[k].input_case_b = j;
                sort_array[k].distance_a_to_b =
                find_distance_1(attribute_rec_out[i].attribute_record_in.subj_attributes,
                        attribute_rec_out[j].attribute_record_in.subj_attributes);
                ++k;
                }
        }

// Sort the sort array

qsort(sort_array, LIMIT2, sizeof(cluster_record), (int(*)(const void *,const void *))sort_function_1);

// Perform the clustering

for (k=0; k<LIMIT2; k++)
        {
        if ((strcmp(attribute_rec_out[sort_array[k].input_case_a].assigned_cluster_name, "DUMMY") == 0)
        && (strcmp(attribute_rec_out[sort_array[k].input_case_b].assigned_cluster_name, "DUMMY") == 0))
                {
                // put both into a new cluster
                new_name(buffer, &new_clust);
                strcpy(attribute_rec_out[sort_array[k].input_case_a].assigned_cluster_name, buffer);
                strcpy(attribute_rec_out[sort_array[k].input_case_b].assigned_cluster_name, buffer);
                }
        if ((strcmp(attribute_rec_out[sort_array[k].input_case_a].assigned_cluster_name, "DUMMY") == 0)
        && (strcmp(attribute_rec_out[sort_array[k].input_case_b].assigned_cluster_name, "DUMMY") != 0))
                   {
                   // one in cluster one not so put the other in the same cluster
                   strcpy(attribute_rec_out[sort_array[k].input_case_a].assigned_cluster_name,
                   attribute_rec_out[sort_array[k].input_case_b].assigned_cluster_name);
                   }
        if ((strcmp(attribute_rec_out[sort_array[k].input_case_a].assigned_cluster_name, "DUMMY") != 0)
        && (strcmp(attribute_rec_out[sort_array[k].input_case_b].assigned_cluster_name, "DUMMY") == 0))
                   {
                   // one in cluster one not so put the other in the same cluster
                   strcpy(attribute_rec_out[sort_array[k].input_case_b].assigned_cluster_name,
                   attribute_rec_out[sort_array[k].input_case_a].assigned_cluster_name);
                   }
        if ((strcmp(attribute_rec_out[sort_array[k].input_case_a].assigned_cluster_name, "DUMMY") != 0)
        && (strcmp(attribute_rec_out[sort_array[k].input_case_b].assigned_cluster_name, "DUMMY") != 0)
        && (strcmp(attribute_rec_out[sort_array[k].input_case_a].assigned_cluster_name,
           attribute_rec_out[sort_array[k].input_case_b].assigned_cluster_name) != 0)
        && (nearly_central(sort_array[k].input_case_a,
                        attribute_rec_out[sort_array[k].input_case_a].assigned_cluster_name,
                        sort_array,
                        attribute_rec_out))
        && (nearly_central(sort_array[k].input_case_b,
                        attribute_rec_out[sort_array[k].input_case_b].assigned_cluster_name,
                        sort_array,
                        attribute_rec_out))
        && ((float)sort_array[k].distance_a_to_b <
(((float)(find_cluster_diameter(attribute_rec_out[sort_array[k].input_case_a].assigned_cluster_name,
sort_array, attribute_rec_out) +

find_cluster_diameter(attribute_rec_out[sort_array[k].input_case_b].assigned_cluster_name, sort_array,
attribute_rec_out)) / 2))))
                   {
                   // both in clusters but different ones
                   new_name(buffer, &new_clust);
                   strcpy(temp_a,attribute_rec_out[sort_array[k].input_case_a].assigned_cluster_name);
                   strcpy(temp_b,attribute_rec_out[sort_array[k].input_case_b].assigned_cluster_name);
                   for (i=0; i<LIMIT1; ++i)
                        {
                        if ((strcmp(attribute_rec_out[i].assigned_cluster_name,temp_a) == 0)
                        || (strcmp(attribute_rec_out[i].assigned_cluster_name,temp_b) == 0))
                                strcpy(attribute_rec_out[i].assigned_cluster_name,buffer);
                        }
                   }
        }

// Sort the output records

qsort(attribute_rec_out, LIMIT1, sizeof(attribute_record_out), (int(*)(const void *,const void
*))sort_function_2);

// Write the clustered records to the output file

fprintf(ofp, "clust00subj000 a001 a002 a003 a004 a005 a006 a007 a008 a009 a010 ");
fprintf(ofp, "a011 a012 a013 a014 a015 a016 a017 a018 a019 a020 a021 a022 a023 a024 ");
fprintf(ofp, "a025 a026 a027 a028 a029 a030 a031 a032 a033 a034 a035 a036 a037 a038 ");
fprintf(ofp, "a039 a040 a041 a042 a043 a044 a045 a046 a047 a048 a049 a050 a051 a052 ");
fprintf(ofp, "a053 a054 a055 a056 a057 a058 a059 a060 a061 a062 a063 a064 a065 a066 ");
fprintf(ofp, "a067 a068 a069 a070 a071 a072 a073 a074 a075 a076 a077 a078 a079 a080 ");
fprintf(ofp, "a081 a082 a083 a084 a085 a086 a087 a088 a089 a090 a091 a092 a093 a094 ");
fprintf(ofp, "a095 a096 a097 a098 a099 a100 a101 a102 a103 a104 a105 a106 a107 a108 ");
fprintf(ofp, "a109 a110 a111 a112 a113 a114 a115 a116 a117 a118 a119 a120 a121 a122 ");
fprintf(ofp, "a123 a124 a125 a126 a127 a128 a129 a130 a131 a132 a133 a134 a135 a136 ");
fprintf(ofp, "a137 a138 a139 a140     *\n");

strcpy(temp_a,"1234567");
```

```
k = 0;           // the current cluster number (reuse k)
j = 0;           // the number of lines in cluster (reuse j)
lines = 0;       // the number of lines written for cluster
max = 0;         // the number of lines in the largest cluster so far
biggest = 0;     // the number of the biggest cluster so far
actual = 0;      // the number of the current cluster
latest = 0;      // the cluster containing the most recent page
cur_weight = 0;  // the current cluster weight
max_weight = 0;  // the largest cluster weight
heaviest = 0;    // the cluster with greatest weight

for (i=0; i<LIMIT1; ++i)
        {
        if (strcmp(attribute_rec_out[i].assigned_cluster_name,temp_a) != 0)
                {
                strcpy(temp_a, attribute_rec_out[i].assigned_cluster_name);

                actual++;             // increment for each cluster processed

                cur_weight = 0;       // reset for a new cluster

                lines = 0;            // reset for new cluster
                }

        lines++;                      // increment for each line written

        if (lines > max)
                {
                max = lines;                // update max
                biggest = actual;           // store for return
                }

        cur_weight = cur_weight + attribute_rec_out[i].weight;

        if (cur_weight > max_weight)
                {
                max_weight = cur_weight;    // update max
                heaviest = actual;          // save heaviest so far
                }

        //printf("Cluster weight %d \n", cur_weight);
        //getche();

        if (strcmp(attribute_rec_out[i].attribute_record_in.subj_id,last_page) == 0)
                {
                latest = actual;      // latest is the cluster position NOT no
                }

        fprintf(ofp, "%s%s    %d    %d    %d    %d    %d    %d    ",
                   attribute_rec_out[i].assigned_cluster_name,
                   attribute_rec_out[i].attribute_record_in.subj_id,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute01,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute02,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute03,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute04,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute05,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute06);

        fprintf(ofp, "%d    %d    %d    %d    %d    %d    %d    %d    ",
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute07,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute08,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute09,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute10,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute11,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute12,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute13,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute14);

        fprintf(ofp, "%d    %d    %d    %d    %d    %d    %d    ",
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute15,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute16,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute17,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute18,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute19,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute20,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute21);

        fprintf(ofp, "%d    %d    %d    %d    %d    %d    %d    ",
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute22,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute23,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute24,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute25,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute26,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute27,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute28);

        fprintf(ofp, "%d    %d    %d    %d    %d    %d    %d    ",
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute29,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute30,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute31,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute32,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute33,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute34,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute35);

         fprintf(ofp, "%d    %d    %d    %d    %d    %d    %d    ",
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute36,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute37,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute38,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute39,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute40,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute41,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute42);

         fprintf(ofp, "%d    %d    %d    %d    %d    %d    %d    ",
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute43,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute44,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute45,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute46,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute47,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute48,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute49);

         fprintf(ofp, "%d    %d    %d    %d    %d    %d    %d    ",
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute50,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute51,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute52,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute53,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute54,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute55,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute56);

         fprintf(ofp, "%d    %d    %d    %d    %d    %d    %d    ",
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute57,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute58,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute59,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute60,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute61,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute62,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute63);

         fprintf(ofp, "%d    %d    %d    %d    %d    %d    %d    ",
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute64,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute65,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute66,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute67,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute68,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute69,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute70);

         fprintf(ofp, "%d    %d    %d    %d    %d    %d    %d    ",
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute71,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute72,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute73,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute74,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute75,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute76,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute77);

         fprintf(ofp, "%d    %d    %d    %d    %d    %d    %d    ",
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute78,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute79,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute80,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute81,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute82,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute83,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute84);

         fprintf(ofp, "%d    %d    %d    %d    %d    %d    %d    ",
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute85,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute86,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute87,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute88,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute89,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute90,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute91);

         fprintf(ofp, "%d    %d    %d    %d    %d    %d    %d    ",
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute92,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute93,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute94,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute95,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute96,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute97,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute98);

         fprintf(ofp, "%d    %d    %d    %d    %d    %d    %d    ",
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute99,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute100,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute101,
                   attribute_rec_out[i].attribute_record_in.subj_attributes.attribute102,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute103,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute104,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute105);

        fprintf(ofp, "%d    %d    %d    %d    %d    %d    %d    ",
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute106,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute107,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute108,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute109,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute110,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute111,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute112);

        fprintf(ofp, "%d    %d    %d    %d    %d    %d    %d    ",
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute113,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute114,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute115,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute116,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute117,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute118,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute119);

        fprintf(ofp, "%d    %d    %d    %d    %d    %d    %d    ",
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute120,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute121,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute122,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute123,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute124,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute125,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute126);

        fprintf(ofp, "%d    %d    %d    %d    %d    %d    %d    ",
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute127,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute128,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute129,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute130,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute131,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute132,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute133);

        fprintf(ofp, "%d    %d    %d    %d    %d    %d    %d xxx ",
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute134,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute135,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute136,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute137,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute138,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute139,
                    attribute_rec_out[i].attribute_record_in.subj_attributes.attribute140);


        if (i == LIMIT1 - 1)
                {
                fprintf(ofp, "**\n");
                }
        else
                {
                fprintf(ofp, "*\n");
                }

        }

fclose(ifp);
fclose(ofp);
fclose(afp);

//printf("Weights : %d %d \n", heaviest, max_weight);
//printf("Largest : %d %d \n", biggest, max);
//printf("Latest  : %d \n", latest);

//getche();

//return actual;     // this is the number of clusters
//return biggest;    // this is the position of the biggest cluster
//return latest;     // this is the position of the cluster with the last page in it
return heaviest;     // this is the position of the heaviest cluster
}

// Fuction to determine if a record is nearly central in a cluster.

int nearly_central(int record_no, char * cluster_name, cluster_record dist_array[], attribute_record_out
output_records[])
{
int find_cluster_diameter(char *, cluster_record *, attribute_record_out *);

int diameter, k;
float delta;

diameter = find_cluster_diameter(cluster_name, dist_array, output_records);
delta = (float)diameter * 0.666;

for (k=0; k<LIMIT2; ++k)
```

```
                    {
                    if ((((dist_array[k].input_case_a == record_no)
                    && (strcmp(output_records[dist_array[k].input_case_a].assigned_cluster_name,
                             output_records[dist_array[k].input_case_b].assigned_cluster_name) == 0))

                    || ((dist_array[k].input_case_b == record_no)
                    && (strcmp(output_records[dist_array[k].input_case_b].assigned_cluster_name,
                             output_records[dist_array[k].input_case_a].assigned_cluster_name) == 0)))

                    && ((float)dist_array[k].distance_a_to_b > delta))
                                            return(0);
                    }
return(1);
}

// Find diameter of the requested cluster.

int find_cluster_diameter(char * cluster_name, cluster_record dist_array[], attribute_record_out
output_records[])
{
int k, max_dist = -1;

for (k=0; k<LIMIT2; k++)
                    {
                    if ((strcmp(output_records[dist_array[k].input_case_a].assigned_cluster_name,
cluster_name) == 0)
                    && (strcmp(output_records[dist_array[k].input_case_b].assigned_cluster_name,
cluster_name) == 0)
                    && (dist_array[k].distance_a_to_b > max_dist))
                                            max_dist = dist_array[k].distance_a_to_b;
                    }
return(max_dist);
}

// Generate a new name for a cluster.

void new_name(char *buffer, int *value)
{
strcpy(buffer, "clust");
sprintf(buffer+5, "%02d", *value);
(*value)++;
}

// This function searches the attribute database and returns the
// attribute description for the supplied subj id.

attribute_data find_subj_attribute_vector(char * search_key, FILE *fp)
{
attribute_file_record afr;
char att_rec[198];              // change to length of the attribute record
int pos;

pos = atoi(search_key+4);
fseek(fp, (pos * 199), 0);      // change to match length of attribute record + 1

fscanf(fp, "%s", att_rec);

att_rec[147] = '\0';
afr.subj_attributes.attribute140 = atoi(att_rec+146);
att_rec[146] = '\0';
afr.subj_attributes.attribute139 = atoi(att_rec+145);
att_rec[145] = '\0';
afr.subj_attributes.attribute138 = atoi(att_rec+144);
att_rec[144] = '\0';
afr.subj_attributes.attribute137 = atoi(att_rec+143);
att_rec[143] = '\0';
afr.subj_attributes.attribute136 = atoi(att_rec+142);
att_rec[142] = '\0';
afr.subj_attributes.attribute135 = atoi(att_rec+141);
att_rec[141] = '\0';
afr.subj_attributes.attribute134 = atoi(att_rec+140);
att_rec[140] = '\0';
afr.subj_attributes.attribute133 = atoi(att_rec+139);
att_rec[139] = '\0';
afr.subj_attributes.attribute132 = atoi(att_rec+138);
att_rec[138] = '\0';
afr.subj_attributes.attribute131 = atoi(att_rec+137);
att_rec[137] = '\0';
afr.subj_attributes.attribute130 = atoi(att_rec+136);
att_rec[136] = '\0';
afr.subj_attributes.attribute129 = atoi(att_rec+135);
att_rec[135] = '\0';
afr.subj_attributes.attribute128 = atoi(att_rec+134);
att_rec[134] = '\0';
afr.subj_attributes.attribute127 = atoi(att_rec+133);
att_rec[133] = '\0';
afr.subj_attributes.attribute126 = atoi(att_rec+132);
att_rec[132] = '\0';
afr.subj_attributes.attribute125 = atoi(att_rec+131);
att_rec[131] = '\0';
afr.subj_attributes.attribute124 = atoi(att_rec+130);
```

```
att_rec[130] = '\0';
afr.subj_attributes.attribute123 = atoi(att_rec+129);
att_rec[129] = '\0';
afr.subj_attributes.attribute122 = atoi(att_rec+128);
att_rec[128] = '\0';
afr.subj_attributes.attribute121 = atoi(att_rec+127);
att_rec[127] = '\0';
afr.subj_attributes.attribute120 = atoi(att_rec+126);
att_rec[126] = '\0';
afr.subj_attributes.attribute119 = atoi(att_rec+125);
att_rec[125] = '\0';
afr.subj_attributes.attribute118 = atoi(att_rec+124);
att_rec[124] = '\0';
afr.subj_attributes.attribute117 = atoi(att_rec+123);
att_rec[123] = '\0';
afr.subj_attributes.attribute116 = atoi(att_rec+122);
att_rec[122] = '\0';
afr.subj_attributes.attribute115 = atoi(att_rec+121);
att_rec[121] = '\0';
afr.subj_attributes.attribute114 = atoi(att_rec+120);
att_rec[120] = '\0';
afr.subj_attributes.attribute113 = atoi(att_rec+119);
att_rec[119] = '\0';
afr.subj_attributes.attribute112 = atoi(att_rec+118);
att_rec[118] = '\0';
afr.subj_attributes.attribute111 = atoi(att_rec+117);
att_rec[117] = '\0';
afr.subj_attributes.attribute110 = atoi(att_rec+116);
att_rec[116] = '\0';
afr.subj_attributes.attribute109 = atoi(att_rec+115);
att_rec[115] = '\0';
afr.subj_attributes.attribute108 = atoi(att_rec+114);
att_rec[114] = '\0';
afr.subj_attributes.attribute107 = atoi(att_rec+113);
att_rec[113] = '\0';
afr.subj_attributes.attribute106 = atoi(att_rec+112);
att_rec[112] = '\0';
afr.subj_attributes.attribute105 = atoi(att_rec+111);
att_rec[111] = '\0';
afr.subj_attributes.attribute104 = atoi(att_rec+110);
att_rec[110] = '\0';
afr.subj_attributes.attribute103 = atoi(att_rec+109);
att_rec[109] = '\0';
afr.subj_attributes.attribute102 = atoi(att_rec+108);
att_rec[108] = '\0';
afr.subj_attributes.attribute101 = atoi(att_rec+107);
att_rec[107] = '\0';
afr.subj_attributes.attribute100 = atoi(att_rec+106);
att_rec[106] = '\0';
afr.subj_attributes.attribute99 = atoi(att_rec+105);
att_rec[105] = '\0';
afr.subj_attributes.attribute98 = atoi(att_rec+104);
att_rec[104] = '\0';
afr.subj_attributes.attribute97 = atoi(att_rec+103);
att_rec[103] = '\0';
afr.subj_attributes.attribute96 = atoi(att_rec+102);
att_rec[102] = '\0';
afr.subj_attributes.attribute95 = atoi(att_rec+101);
att_rec[101] = '\0';
afr.subj_attributes.attribute94 = atoi(att_rec+100);
att_rec[100] = '\0';
afr.subj_attributes.attribute93 = atoi(att_rec+99);
att_rec[99] = '\0';
afr.subj_attributes.attribute92 = atoi(att_rec+98);
att_rec[98] = '\0';
afr.subj_attributes.attribute91 = atoi(att_rec+97);
att_rec[97] = '\0';
afr.subj_attributes.attribute90 = atoi(att_rec+96);
att_rec[96] = '\0';
afr.subj_attributes.attribute89 = atoi(att_rec+95);
att_rec[95] = '\0';
afr.subj_attributes.attribute88 = atoi(att_rec+94);
att_rec[94] = '\0';
afr.subj_attributes.attribute87 = atoi(att_rec+93);
att_rec[93] = '\0';
afr.subj_attributes.attribute86 = atoi(att_rec+92);
att_rec[92] = '\0';
afr.subj_attributes.attribute85 = atoi(att_rec+91);
att_rec[91] = '\0';
afr.subj_attributes.attribute84 = atoi(att_rec+90);
att_rec[90] = '\0';
afr.subj_attributes.attribute83 = atoi(att_rec+89);
att_rec[89] = '\0';
afr.subj_attributes.attribute82 = atoi(att_rec+88);
att_rec[88] = '\0';
afr.subj_attributes.attribute81 = atoi(att_rec+87);
att_rec[87] = '\0';
afr.subj_attributes.attribute80 = atoi(att_rec+86);
att_rec[86] = '\0';
afr.subj_attributes.attribute79 = atoi(att_rec+85);
att_rec[85] = '\0';
```

```
afr.subj_attributes.attribute78 = atoi(att_rec+84);
att_rec[84] = '\0';
afr.subj_attributes.attribute77 = atoi(att_rec+83);
att_rec[83] = '\0';
afr.subj_attributes.attribute76 = atoi(att_rec+82);
att_rec[82] = '\0';
afr.subj_attributes.attribute75 = atoi(att_rec+81);
att_rec[81] = '\0';
afr.subj_attributes.attribute74 = atoi(att_rec+80);
att_rec[80] = '\0';
afr.subj_attributes.attribute73 = atoi(att_rec+79);
att_rec[79] = '\0';
afr.subj_attributes.attribute72 = atoi(att_rec+78);
att_rec[78] = '\0';
afr.subj_attributes.attribute71 = atoi(att_rec+77);
att_rec[77] = '\0';
afr.subj_attributes.attribute70 = atoi(att_rec+76);
att_rec[76] = '\0';
afr.subj_attributes.attribute69 = atoi(att_rec+75);
att_rec[75] = '\0';
afr.subj_attributes.attribute68 = atoi(att_rec+74);
att_rec[74] = '\0';
afr.subj_attributes.attribute67 = atoi(att_rec+73);
att_rec[73] = '\0';
afr.subj_attributes.attribute66 = atoi(att_rec+72);
att_rec[72] = '\0';
afr.subj_attributes.attribute65 = atoi(att_rec+71);
att_rec[71] = '\0';
afr.subj_attributes.attribute64 = atoi(att_rec+70);
att_rec[70] = '\0';
afr.subj_attributes.attribute63 = atoi(att_rec+69);
att_rec[69] = '\0';
afr.subj_attributes.attribute62 = atoi(att_rec+68);
att_rec[68] = '\0';
afr.subj_attributes.attribute61 = atoi(att_rec+67);
att_rec[67] = '\0';
afr.subj_attributes.attribute60 = atoi(att_rec+66);
att_rec[66] = '\0';
afr.subj_attributes.attribute59 = atoi(att_rec+65);
att_rec[65] = '\0';
afr.subj_attributes.attribute58 = atoi(att_rec+64);
att_rec[64] = '\0';
afr.subj_attributes.attribute57 = atoi(att_rec+63);
att_rec[63] = '\0';
afr.subj_attributes.attribute56 = atoi(att_rec+62);
att_rec[62] = '\0';
afr.subj_attributes.attribute55 = atoi(att_rec+61);
att_rec[61] = '\0';
afr.subj_attributes.attribute54 = atoi(att_rec+60);
att_rec[60] = '\0';
afr.subj_attributes.attribute53 = atoi(att_rec+59);
att_rec[59] = '\0';
afr.subj_attributes.attribute52 = atoi(att_rec+58);
att_rec[58] = '\0';
afr.subj_attributes.attribute51 = atoi(att_rec+57);
att_rec[57] = '\0';
afr.subj_attributes.attribute50 = atoi(att_rec+56);
att_rec[56] = '\0';
afr.subj_attributes.attribute49 = atoi(att_rec+55);
att_rec[55] = '\0';
afr.subj_attributes.attribute48 = atoi(att_rec+54);
att_rec[54] = '\0';
afr.subj_attributes.attribute47 = atoi(att_rec+53);
att_rec[53] = '\0';
afr.subj_attributes.attribute46 = atoi(att_rec+52);
att_rec[52] = '\0';
afr.subj_attributes.attribute45 = atoi(att_rec+51);
att_rec[51] = '\0';
afr.subj_attributes.attribute44 = atoi(att_rec+50);
att_rec[50] = '\0';
afr.subj_attributes.attribute43 = atoi(att_rec+49);
att_rec[49] = '\0';
afr.subj_attributes.attribute42 = atoi(att_rec+48);
att_rec[48] = '\0';
afr.subj_attributes.attribute41 = atoi(att_rec+47);
att_rec[47] = '\0';
afr.subj_attributes.attribute40 = atoi(att_rec+46);
att_rec[46] = '\0';
afr.subj_attributes.attribute39 = atoi(att_rec+45);
att_rec[45] = '\0';
afr.subj_attributes.attribute38 = atoi(att_rec+44);
att_rec[44] = '\0';
afr.subj_attributes.attribute37 = atoi(att_rec+43);
att_rec[43] = '\0';
afr.subj_attributes.attribute36 = atoi(att_rec+42);
att_rec[42] = '\0';
afr.subj_attributes.attribute35 = atoi(att_rec+41);
att_rec[41] = '\0';
afr.subj_attributes.attribute34 = atoi(att_rec+40);
att_rec[40] = '\0';
afr.subj_attributes.attribute33 = atoi(att_rec+39);
```

```
att_rec[39] = '\0';
afr.subj_attributes.attribute32 = atoi(att_rec+38);
att_rec[38] = '\0';
afr.subj_attributes.attribute31 = atoi(att_rec+37);
att_rec[37] = '\0';
afr.subj_attributes.attribute30 = atoi(att_rec+36);
att_rec[36] = '\0';
afr.subj_attributes.attribute29 = atoi(att_rec+35);
att_rec[35] = '\0';
afr.subj_attributes.attribute28 = atoi(att_rec+34);
att_rec[34] = '\0';
afr.subj_attributes.attribute27 = atoi(att_rec+33);
att_rec[33] = '\0';
afr.subj_attributes.attribute26 = atoi(att_rec+32);
att_rec[32] = '\0';
afr.subj_attributes.attribute25 = atoi(att_rec+31);
att_rec[31] = '\0';
afr.subj_attributes.attribute24 = atoi(att_rec+30);
att_rec[30] = '\0';
afr.subj_attributes.attribute23 = atoi(att_rec+29);
att_rec[29] = '\0';
afr.subj_attributes.attribute22 = atoi(att_rec+28);
att_rec[28] = '\0';
afr.subj_attributes.attribute21 = atoi(att_rec+27);
att_rec[27] = '\0';
afr.subj_attributes.attribute20 = atoi(att_rec+26);
att_rec[26] = '\0';
afr.subj_attributes.attribute19 = atoi(att_rec+25);
att_rec[25] = '\0';
afr.subj_attributes.attribute18 = atoi(att_rec+24);
att_rec[24] = '\0';
afr.subj_attributes.attribute17 = atoi(att_rec+23);
att_rec[23] = '\0';
afr.subj_attributes.attribute16 = atoi(att_rec+22);
att_rec[22] = '\0';
afr.subj_attributes.attribute15 = atoi(att_rec+21);
att_rec[21] = '\0';
afr.subj_attributes.attribute14 = atoi(att_rec+20);
att_rec[20] = '\0';
afr.subj_attributes.attribute13 = atoi(att_rec+19);
att_rec[19] = '\0';
afr.subj_attributes.attribute12 = atoi(att_rec+18);
att_rec[18] = '\0';
afr.subj_attributes.attribute11 = atoi(att_rec+17);
att_rec[17] = '\0';
afr.subj_attributes.attribute10 = atoi(att_rec+16);
att_rec[16] = '\0';
afr.subj_attributes.attribute09 = atoi(att_rec+15);
att_rec[15] = '\0';
afr.subj_attributes.attribute08 = atoi(att_rec+14);
att_rec[14] = '\0';
afr.subj_attributes.attribute07 = atoi(att_rec+13);
att_rec[13] = '\0';
afr.subj_attributes.attribute06 = atoi(att_rec+12);
att_rec[12] = '\0';
afr.subj_attributes.attribute05 = atoi(att_rec+11);
att_rec[11] = '\0';
afr.subj_attributes.attribute04 = atoi(att_rec+10);
att_rec[10] = '\0';
afr.subj_attributes.attribute03 = atoi(att_rec+9);
att_rec[9] = '\0';
afr.subj_attributes.attribute02 = atoi(att_rec+8);
att_rec[8] = '\0';
afr.subj_attributes.attribute01 = atoi(att_rec+7);
att_rec[7] = '\0';
strcpy(afr.subj_id, att_rec);

if (strcmp(afr.subj_id, search_key) == 0)
                   return (afr.subj_attributes);

printf ("Attribute not found for key!!! %s %d \n", search_key, pos);

return (afr.subj_attributes); // to prevent compiler warning only
}

// This is the cluster QSORT sort function.

int sort_function_1(cluster_record *first, cluster_record *second)
{
        if (first->distance_a_to_b < second->distance_a_to_b)
                   return (-1);
        else if (first->distance_a_to_b > second->distance_a_to_b)
                   return (+1);
        else
                   return (0);
}

// This is the output QSORT sort function.

int sort_function_2(attribute_record_out *first, attribute_record_out *second)
{
        return(strcmp(first->assigned_cluster_name, second->assigned_cluster_name));
}

// This function finds the distance between two attribute records. The
// pseudo metric employed is the number of differing attribute values.

int find_distance_1(attribute_data rec01, attribute_data rec02)
{
int distance = 0;

if (rec01.attribute01 != rec02.attribute01)
                  ++distance;

if (rec01.attribute02 != rec02.attribute02)
                  ++distance;

if (rec01.attribute03 != rec02.attribute03)
                  ++distance;

if (rec01.attribute04 != rec02.attribute04)
                  ++distance;

if (rec01.attribute05 != rec02.attribute05)
                  ++distance;

if (rec01.attribute06 != rec02.attribute06)
                  ++distance;

if (rec01.attribute07 != rec02.attribute07)
                  ++distance;

if (rec01.attribute08 != rec02.attribute08)
                  ++distance;

if (rec01.attribute09 != rec02.attribute09)
                  ++distance;

if (rec01.attribute10 != rec02.attribute10)
                  ++distance;

if (rec01.attribute11 != rec02.attribute11)
                  ++distance;

if (rec01.attribute12 != rec02.attribute12)
                  ++distance;

if (rec01.attribute13 != rec02.attribute13)
                  ++distance;

if (rec01.attribute14 != rec02.attribute14)
                  ++distance;

if (rec01.attribute15 != rec02.attribute15)
                  ++distance;

if (rec01.attribute16 != rec02.attribute16)
                  ++distance;

if (rec01.attribute17 != rec02.attribute17)
                  ++distance;

if (rec01.attribute18 != rec02.attribute18)
                  ++distance;

if (rec01.attribute19 != rec02.attribute19)
                  ++distance;

if (rec01.attribute20 != rec02.attribute20)
                  ++distance;

if (rec01.attribute21 != rec02.attribute21)
                  ++distance;

if (rec01.attribute22 != rec02.attribute22)
                  ++distance;

if (rec01.attribute23 != rec02.attribute23)
                  ++distance;

if (rec01.attribute24 != rec02.attribute24)
                  ++distance;

if (rec01.attribute25 != rec02.attribute25)
                  ++distance;

if (rec01.attribute26 != rec02.attribute26)
                  ++distance;

if (rec01.attribute27 != rec02.attribute27)
                  ++distance;
```

```
if (rec01.attribute28 != rec02.attribute28)
                   ++distance;

if (rec01.attribute29 != rec02.attribute29)
                   ++distance;

if (rec01.attribute30 != rec02.attribute30)
                   ++distance;

if (rec01.attribute31 != rec02.attribute31)
                   ++distance;

if (rec01.attribute32 != rec02.attribute32)
                   ++distance;

if (rec01.attribute33 != rec02.attribute33)
                   ++distance;

if (rec01.attribute34 != rec02.attribute34)
                   ++distance;

if (rec01.attribute35 != rec02.attribute35)
                   ++distance;

if (rec01.attribute36 != rec02.attribute36)
                   ++distance;

if (rec01.attribute37 != rec02.attribute37)
                   ++distance;

if (rec01.attribute38 != rec02.attribute38)
                   ++distance;

if (rec01.attribute39 != rec02.attribute39)
                   ++distance;

if (rec01.attribute40 != rec02.attribute40)
                   ++distance;

if (rec01.attribute41 != rec02.attribute41)
                   ++distance;

if (rec01.attribute42 != rec02.attribute42)
                   ++distance;

if (rec01.attribute43 != rec02.attribute43)
                   ++distance;

if (rec01.attribute44 != rec02.attribute44)
                   ++distance;

if (rec01.attribute45 != rec02.attribute45)
                   ++distance;

if (rec01.attribute46 != rec02.attribute46)
                   ++distance;

if (rec01.attribute47 != rec02.attribute47)
                   ++distance;

if (rec01.attribute48 != rec02.attribute48)
                   ++distance;

if (rec01.attribute49 != rec02.attribute49)
                   ++distance;

if (rec01.attribute50 != rec02.attribute50)
                   ++distance;

if (rec01.attribute51 != rec02.attribute51)
                   ++distance;

if (rec01.attribute52 != rec02.attribute52)
                   ++distance;

if (rec01.attribute53 != rec02.attribute53)
                   ++distance;

if (rec01.attribute54 != rec02.attribute54)
                   ++distance;

if (rec01.attribute55 != rec02.attribute55)
                   ++distance;

if (rec01.attribute56 != rec02.attribute56)
                   ++distance;

if (rec01.attribute57 != rec02.attribute57)
                   ++distance;

if (rec01.attribute58 != rec02.attribute58)
```

```
                   ++distance;

if (rec01.attribute59 != rec02.attribute59)
                   ++distance;

if (rec01.attribute60 != rec02.attribute60)
                   ++distance;

if (rec01.attribute61 != rec02.attribute61)
                   ++distance;

if (rec01.attribute62 != rec02.attribute62)
                   ++distance;

if (rec01.attribute63 != rec02.attribute63)
                   ++distance;

if (rec01.attribute64 != rec02.attribute64)
                   ++distance;

if (rec01.attribute65 != rec02.attribute65)
                   ++distance;

if (rec01.attribute66 != rec02.attribute66)
                   ++distance;

if (rec01.attribute67 != rec02.attribute67)
                   ++distance;

if (rec01.attribute68 != rec02.attribute68)
                   ++distance;

if (rec01.attribute69 != rec02.attribute69)
                   ++distance;

if (rec01.attribute70 != rec02.attribute70)
                   ++distance;

if (rec01.attribute71 != rec02.attribute71)
                   ++distance;

if (rec01.attribute72 != rec02.attribute72)
                   ++distance;

if (rec01.attribute73 != rec02.attribute73)
                   ++distance;

if (rec01.attribute74 != rec02.attribute74)
                   ++distance;

if (rec01.attribute75 != rec02.attribute75)
                   ++distance;

if (rec01.attribute76 != rec02.attribute76)
                   ++distance;

if (rec01.attribute77 != rec02.attribute77)
                   ++distance;

if (rec01.attribute78 != rec02.attribute78)
                   ++distance;

if (rec01.attribute79 != rec02.attribute79)
                   ++distance;

if (rec01.attribute80 != rec02.attribute80)
                   ++distance;

if (rec01.attribute81 != rec02.attribute81)
                   ++distance;

if (rec01.attribute82 != rec02.attribute82)
                   ++distance;

if (rec01.attribute83 != rec02.attribute83)
                   ++distance;

if (rec01.attribute84 != rec02.attribute84)
                   ++distance;

if (rec01.attribute85 != rec02.attribute85)
                   ++distance;

if (rec01.attribute86 != rec02.attribute86)
                   ++distance;

if (rec01.attribute87 != rec02.attribute87)
                   ++distance;

if (rec01.attribute88 != rec02.attribute88)
                   ++distance;
```

```
if (rec01.attribute89 != rec02.attribute89)
                    ++distance;

if (rec01.attribute90 != rec02.attribute90)
                    ++distance;

if (rec01.attribute91 != rec02.attribute91)
                    ++distance;

if (rec01.attribute92 != rec02.attribute92)
                    ++distance;

if (rec01.attribute93 != rec02.attribute93)
                    ++distance;

if (rec01.attribute94 != rec02.attribute94)
                    ++distance;

if (rec01.attribute95 != rec02.attribute95)
                    ++distance;

if (rec01.attribute96 != rec02.attribute96)
                    ++distance;

if (rec01.attribute97 != rec02.attribute97)
                    ++distance;

if (rec01.attribute98 != rec02.attribute98)
                    ++distance;

if (rec01.attribute99 != rec02.attribute99)
                    ++distance;

if (rec01.attribute100 != rec02.attribute100)
                    ++distance;

if (rec01.attribute101 != rec02.attribute101)
                    ++distance;

if (rec01.attribute102 != rec02.attribute102)
                    ++distance;

if (rec01.attribute103 != rec02.attribute103)
                    ++distance;

if (rec01.attribute104 != rec02.attribute104)
                    ++distance;

if (rec01.attribute105 != rec02.attribute105)
                    ++distance;

if (rec01.attribute106 != rec02.attribute106)
                    ++distance;

if (rec01.attribute107 != rec02.attribute107)
                    ++distance;

if (rec01.attribute108 != rec02.attribute108)
                    ++distance;

if (rec01.attribute109 != rec02.attribute109)
                    ++distance;

if (rec01.attribute110 != rec02.attribute110)
                    ++distance;

if (rec01.attribute111 != rec02.attribute111)
                    ++distance;

if (rec01.attribute112 != rec02.attribute112)
                    ++distance;

if (rec01.attribute113 != rec02.attribute113)
                    ++distance;

if (rec01.attribute114 != rec02.attribute114)
                    ++distance;

if (rec01.attribute115 != rec02.attribute115)
                    ++distance;

if (rec01.attribute116 != rec02.attribute116)
                    ++distance;

if (rec01.attribute117 != rec02.attribute117)
                    ++distance;

if (rec01.attribute118 != rec02.attribute118)
                    ++distance;
```

```
if (rec01.attribute119 != rec02.attribute119)
                   ++distance;

if (rec01.attribute120 != rec02.attribute120)
                   ++distance;

if (rec01.attribute121 != rec02.attribute121)
                   ++distance;

if (rec01.attribute122 != rec02.attribute122)
                   ++distance;

if (rec01.attribute123 != rec02.attribute123)
                   ++distance;

if (rec01.attribute124 != rec02.attribute124)
                   ++distance;

if (rec01.attribute125 != rec02.attribute125)
                   ++distance;

if (rec01.attribute126 != rec02.attribute126)
                   ++distance;

if (rec01.attribute127 != rec02.attribute127)
                   ++distance;

if (rec01.attribute128 != rec02.attribute128)
                   ++distance;

if (rec01.attribute129 != rec02.attribute129)
                   ++distance;

if (rec01.attribute130 != rec02.attribute130)
                   ++distance;

if (rec01.attribute131 != rec02.attribute131)
                   ++distance;

if (rec01.attribute132 != rec02.attribute132)
                   ++distance;

if (rec01.attribute133 != rec02.attribute133)
                   ++distance;

if (rec01.attribute134 != rec02.attribute134)
                   ++distance;

if (rec01.attribute135 != rec02.attribute135)
                   ++distance;

if (rec01.attribute136 != rec02.attribute136)
                   ++distance;

if (rec01.attribute137 != rec02.attribute137)
                   ++distance;

if (rec01.attribute138 != rec02.attribute138)
                   ++distance;

if (rec01.attribute139 != rec02.attribute139)
                   ++distance;

if (rec01.attribute140 != rec02.attribute140)
                   ++distance;

return distance;
}
```