// ROC5.cpp : Defines the entry point for the console application.

//

#include <stdlib.h> /\* For \_MAX\_PATH definition \*/

#include "stdafx.h"

#include <stdio.h>

#include <math.h>

#include <malloc.h>

/\*#include <iostream.h>\*/

#include <string.h>

#include <time.h>

#include <cstdlib>

/\* Revision 4.0 April 10 2002 edited final EOF on data.txt entry problems \*/

/\* Revision 4.1 July 3 2002 fixed rounding problems \*/

/\* Revision 4.11 July 12 2002 fixed more truncating problems \*/

/\* Revision 4.12 July 31 2002 fixed more truncating problems \*/

/\* Revision 4.13 Oct. 6 2003 capacity increase to deal with Rosenheck data 50000x200\*/

/\* Revision 4.14 Oct. 10 2003 print error repaired; Dx for stdev errors; capacity to 50000x1000\*/

/\* Revision 4.15 Nov. 10 2003 fixed errors in non 50/50 sensitivity/specificity kappa\*/

/\* Revision 4.16 Dec. 24 2003 make sorting and printing intermediate values optional to speed processing\*/

/\* Revision 4.17 Dec. 25 2003 supress output of all diagrams with marginal totals < 10\*/

/\* Revision 4.18 Dec. 28 2003 supress sort of all kappa\*/

/\* Revision 4.19 Jul. 13 2004 fix error in SD of floating point variables\*/

/\* Revision 4.20 Jan. 7 2006 Allow 01 gradation of settings for sens/spec\*/

/\* Revision 4.21 Apr. 7 2006 Allow .05 and .01 criteria\*/

/\* Revision 4.22 May. 25 2007 Allows 120k cases and 1k variables\*/

/\* Revision 4.30 July 5 2007 Allows flexible number cases and variables\*/

/\* Revision 4.32 July 6 2007 Allows flexible number cases and variables memory usuage reduced\*/

/\* Revision 5.00 Sept 26 2007 Option for 64 bit computing of large datasets plus all the rest of upgrades from 4.19 on\*/

/\* This is a major rework designed to handle genetic data\*/

/\* Code for this version with run on 32bit and 64bit machines, but must be compiled for the correct machine\*/

/\* This version will be released precompiled for various processors\*/

/\* Revision 5.00 Aug 21 2008 Version moved from 64-bit Xeno to 32-bit M2300 to compile distributable version\*/

/\* Revision 5.01 Jun 13 2014 5.00 recompiled for Windows 7 and more modern Windows OSs\*/

/\* Revision 5.02 Jun 17 2014 5.00 output diagrams truncated to reduce ambiguities.... W64 and W32 equated to same codebase\*/

/\* Revision 5.03 Sep 30 2014 Marginal cut points all set to =< value entered output format fixed .... W64 and W32 equated to same codebase\*/

/\* Revision 5.04 Jul 1 2015 Minor fixes following UCSF suggestions; Missing values must only be -9999 not -9999.99 .... W64 and W32 equated to same codebase\*/

/\* Revision 5.06 Jul 6 2015 Attempt to get Sens and Spec printed on summary page .... W64 and W32 equated to same codebase\*/

/\* Revision 5.07 Jul 13 2015 Best Variable format changed to f8.3d (3 digits significance) throughout printout .... W64 and W32 equated to same codebase\*/

/\* Revision 5.07 Jul 13 2015 Variable Name length changed to 24 characters throughout printout .... W64 and W32 equated to same codebase\*/

/\* these are the numbers of rows and columns in the array \*/

#define NCOL 600000 /\* data[NCOL][NROW] = data[cases\_max][variables\_max] = data[k][j]\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*OLD NOW UNLIMITED JUNE 2014\*/

#define NROW 800 /\* this will become data[NCOL][NROW] - there are as many stars in\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*OLD NOW UNLIMITED JUNE 2014\*/

/\* the follows, as there are dimensions in the final array (2 in this case) \*/

#define NRUN 15 /\* number of maximum number of runs of program initial (1) second (2-3) third (4-7) fourth (8-15) \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MAJOR MEMORY ALLOCATIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

float \*\*data, \*\*data2;

short \*\*True\_Positives, \*\*True\_Negatives, \*\*False\_Positives, \*\*False\_Negatives;

short \*\*PVP\_denominator, \*\*PVN\_denominator;

\_\_int8 \*\*LT\_GE;

float \*\*PVP, \*\*PVN;

float \*\*chi\_square, \*\*k0\_5;

float p\_value\_to\_progress, chi\_value\_to\_progress;

float \*variables\_sum, \*variables\_mean, \*variables\_n, \*variables\_sumsq, \*variables\_sumrt;

//float variables\_maximum[NCOL],variables\_minimum[NCOL];

float \*variables\_maximum,\*variables\_minimum;

//float best\_value[NRUN][NCOL], best\_kappa[NRUN][NCOL], best\_variable[NRUN][NCOL] best\_chi\_square[NRUN][NCOL];

float \*\*best\_value, \*\*best\_kappa, \*\*best\_variable, \*\*best\_chi\_square, \*\*best\_sensitivity, \*\*best\_specificity, \*\*sensitivity, \*\*specificity;

//float best\_PVP[NRUN][NCOL], best\_PVP\_denominator[NRUN][NCOL], best\_PVN[NRUN][NCOL], best\_PVN\_denominator[NRUN][NCOL], best\_LT\_GE[NRUN][NCOL];

float \*\*best\_PVP, \*\*best\_PVP\_denominator, \*\*best\_PVN, \*\*best\_PVN\_denominator, \*\*best\_LT\_GE;

char \*\*var;

//char var[NCOL][25];

char c, percent\_sign, malstring,string[25],chi\_square\_sig[25];

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MINOR MEMORY ALLOCATIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

long i,j,k,l,m,n,variables,cases,variables2,cases2,letters, cases\_max, variables\_max;

float f, oldf, temp\_data\_ss, datatemp;

float aa,bb,cc,dd,ac,ab,bd,cd,abcd, term1, term2,term3,term4,marginal\_minimum;

float P,P1, Eff, Sp, Se, Q, Q1, k0\_0, k1\_0;

int negative\_switch, kappa\_level, plot, run, run2, run3, print\_long, DE\_BUG,BIT64, j2, k2;

float TP, FP, FN, TN, MISSING\_FLOAT, r, percent\_positivef;

int var\_number, total\_n, percent\_positive, MISSING\_INT, marginal\_minimum\_i;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FUNCTION DEFINITIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double sdndev(double sumsq,double sumrt,double n); /\*New type function definition\*/

void computation(int run);

void zero\_variables();

void separators();

void recover\_data();

void print\_kappa();

void print\_data();

void remove\_GE\_Data(int run2);

void remove\_LT\_Data(int run3);

void dimension\_array\_size();

void print\_variable\_names(int var\_number);

void print\_variable\_names\_short(int var\_number);

void print\_variable\_names\_eight(int var\_number);

void print\_variable\_names\_eight\_blank(int var\_number);

void print\_long\_line();

void print\_summary();

void print\_diagram();

void parse\_command\_line();

FILE \*fp;

int main(int argc, char\* argv[])

{

if (DE\_BUG == 1) printf("0000 COMMAND LINE argc = %d kappa\_level2 = %d plot3 = %d print\_long4 = %d DE\_BUG5 = %d p\_value\_to\_progress6 = %f\n",argc, kappa\_level, plot, print\_long, DE\_BUG, p\_value\_to\_progress);

MISSING\_INT = -9999;

MISSING\_FLOAT = -999999999999.99;

percent\_sign = '%';

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*Crudely Parses command line argumentfor kappa level/ no atoi() in C++? \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*This is crude because the various functions that do this change with versions of C and C++ \*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

kappa\_level = 50;

if( argc > 2 && \_stricmp( argv[2], "0" ) == 0 ) kappa\_level = 0;

if( argc > 2 && \_stricmp( argv[2], "1" ) == 0 ) kappa\_level = 1;

if( argc > 2 && \_stricmp( argv[2], "2" ) == 0 ) kappa\_level = 2;

if( argc > 2 && \_stricmp( argv[2], "3" ) == 0 ) kappa\_level = 3;

if( argc > 2 && \_stricmp( argv[2], "4" ) == 0 ) kappa\_level = 4;

if( argc > 2 && \_stricmp( argv[2], "5" ) == 0 ) kappa\_level = 5;

if( argc > 2 && \_stricmp( argv[2], "6" ) == 0 ) kappa\_level = 6;

if( argc > 2 && \_stricmp( argv[2], "7" ) == 0 ) kappa\_level = 7;

if( argc > 2 && \_stricmp( argv[2], "8" ) == 0 ) kappa\_level = 8;

if( argc > 2 && \_stricmp( argv[2], "9" ) == 0 ) kappa\_level = 9;

if( argc > 2 && \_stricmp( argv[2], "00" ) == 0 ) kappa\_level = 0;

if( argc > 2 && \_stricmp( argv[2], "01" ) == 0 ) kappa\_level = 1;

if( argc > 2 && \_stricmp( argv[2], "02" ) == 0 ) kappa\_level = 2;

if( argc > 2 && \_stricmp( argv[2], "03" ) == 0 ) kappa\_level = 3;

if( argc > 2 && \_stricmp( argv[2], "04" ) == 0 ) kappa\_level = 4;

if( argc > 2 && \_stricmp( argv[2], "05" ) == 0 ) kappa\_level = 5;

if( argc > 2 && \_stricmp( argv[2], "06" ) == 0 ) kappa\_level = 6;

if( argc > 2 && \_stricmp( argv[2], "07" ) == 0 ) kappa\_level = 7;

if( argc > 2 && \_stricmp( argv[2], "08" ) == 0 ) kappa\_level = 8;

if( argc > 2 && \_stricmp( argv[2], "09" ) == 0 ) kappa\_level = 9;

if( argc > 2 && \_stricmp( argv[2], "10" ) == 0 ) kappa\_level = 10;

if( argc > 2 && \_stricmp( argv[2], "11" ) == 0 ) kappa\_level = 11;

if( argc > 2 && \_stricmp( argv[2], "12" ) == 0 ) kappa\_level = 12;

if( argc > 2 && \_stricmp( argv[2], "13" ) == 0 ) kappa\_level = 13;

if( argc > 2 && \_stricmp( argv[2], "14" ) == 0 ) kappa\_level = 14;

if( argc > 2 && \_stricmp( argv[2], "15" ) == 0 ) kappa\_level = 15;

if( argc > 2 && \_stricmp( argv[2], "16" ) == 0 ) kappa\_level = 16;

if( argc > 2 && \_stricmp( argv[2], "17" ) == 0 ) kappa\_level = 17;

if( argc > 2 && \_stricmp( argv[2], "18" ) == 0 ) kappa\_level = 18;

if( argc > 2 && \_stricmp( argv[2], "19" ) == 0 ) kappa\_level = 19;

if( argc > 2 && \_stricmp( argv[2], "20" ) == 0 ) kappa\_level = 20;

if( argc > 2 && \_stricmp( argv[2], "21" ) == 0 ) kappa\_level = 21;

if( argc > 2 && \_stricmp( argv[2], "22" ) == 0 ) kappa\_level = 22;

if( argc > 2 && \_stricmp( argv[2], "23" ) == 0 ) kappa\_level = 23;

if( argc > 2 && \_stricmp( argv[2], "24" ) == 0 ) kappa\_level = 24;

if( argc > 2 && \_stricmp( argv[2], "25" ) == 0 ) kappa\_level = 25;

if( argc > 2 && \_stricmp( argv[2], "26" ) == 0 ) kappa\_level = 26;

if( argc > 2 && \_stricmp( argv[2], "27" ) == 0 ) kappa\_level = 27;

if( argc > 2 && \_stricmp( argv[2], "28" ) == 0 ) kappa\_level = 28;

if( argc > 2 && \_stricmp( argv[2], "29" ) == 0 ) kappa\_level = 29;

if( argc > 2 && \_stricmp( argv[2], "30" ) == 0 ) kappa\_level = 30;

if( argc > 2 && \_stricmp( argv[2], "31" ) == 0 ) kappa\_level = 31;

if( argc > 2 && \_stricmp( argv[2], "32" ) == 0 ) kappa\_level = 32;

if( argc > 2 && \_stricmp( argv[2], "33" ) == 0 ) kappa\_level = 33;

if( argc > 2 && \_stricmp( argv[2], "34" ) == 0 ) kappa\_level = 34;

if( argc > 2 && \_stricmp( argv[2], "35" ) == 0 ) kappa\_level = 35;

if( argc > 2 && \_stricmp( argv[2], "36" ) == 0 ) kappa\_level = 36;

if( argc > 2 && \_stricmp( argv[2], "37" ) == 0 ) kappa\_level = 37;

if( argc > 2 && \_stricmp( argv[2], "38" ) == 0 ) kappa\_level = 38;

if( argc > 2 && \_stricmp( argv[2], "39" ) == 0 ) kappa\_level = 39;

if( argc > 2 && \_stricmp( argv[2], "40" ) == 0 ) kappa\_level = 40;

if( argc > 2 && \_stricmp( argv[2], "41" ) == 0 ) kappa\_level = 41;

if( argc > 2 && \_stricmp( argv[2], "42" ) == 0 ) kappa\_level = 42;

if( argc > 2 && \_stricmp( argv[2], "43" ) == 0 ) kappa\_level = 43;

if( argc > 2 && \_stricmp( argv[2], "44" ) == 0 ) kappa\_level = 44;

if( argc > 2 && \_stricmp( argv[2], "45" ) == 0 ) kappa\_level = 45;

if( argc > 2 && \_stricmp( argv[2], "46" ) == 0 ) kappa\_level = 46;

if( argc > 2 && \_stricmp( argv[2], "47" ) == 0 ) kappa\_level = 47;

if( argc > 2 && \_stricmp( argv[2], "48" ) == 0 ) kappa\_level = 48;

if( argc > 2 && \_stricmp( argv[2], "49" ) == 0 ) kappa\_level = 49;

if( argc > 2 && \_stricmp( argv[2], "50" ) == 0 ) kappa\_level = 50;

if( argc > 2 && \_stricmp( argv[2], "51" ) == 0 ) kappa\_level = 51;

if( argc > 2 && \_stricmp( argv[2], "52" ) == 0 ) kappa\_level = 52;

if( argc > 2 && \_stricmp( argv[2], "53" ) == 0 ) kappa\_level = 53;

if( argc > 2 && \_stricmp( argv[2], "54" ) == 0 ) kappa\_level = 54;

if( argc > 2 && \_stricmp( argv[2], "55" ) == 0 ) kappa\_level = 55;

if( argc > 2 && \_stricmp( argv[2], "56" ) == 0 ) kappa\_level = 56;

if( argc > 2 && \_stricmp( argv[2], "57" ) == 0 ) kappa\_level = 57;

if( argc > 2 && \_stricmp( argv[2], "58" ) == 0 ) kappa\_level = 58;

if( argc > 2 && \_stricmp( argv[2], "59" ) == 0 ) kappa\_level = 59;

if( argc > 2 && \_stricmp( argv[2], "60" ) == 0 ) kappa\_level = 60;

if( argc > 2 && \_stricmp( argv[2], "61" ) == 0 ) kappa\_level = 61;

if( argc > 2 && \_stricmp( argv[2], "62" ) == 0 ) kappa\_level = 62;

if( argc > 2 && \_stricmp( argv[2], "63" ) == 0 ) kappa\_level = 63;

if( argc > 2 && \_stricmp( argv[2], "64" ) == 0 ) kappa\_level = 64;

if( argc > 2 && \_stricmp( argv[2], "65" ) == 0 ) kappa\_level = 65;

if( argc > 2 && \_stricmp( argv[2], "66" ) == 0 ) kappa\_level = 66;

if( argc > 2 && \_stricmp( argv[2], "67" ) == 0 ) kappa\_level = 67;

if( argc > 2 && \_stricmp( argv[2], "68" ) == 0 ) kappa\_level = 68;

if( argc > 2 && \_stricmp( argv[2], "69" ) == 0 ) kappa\_level = 69;

if( argc > 2 && \_stricmp( argv[2], "70" ) == 0 ) kappa\_level = 70;

if( argc > 2 && \_stricmp( argv[2], "71" ) == 0 ) kappa\_level = 71;

if( argc > 2 && \_stricmp( argv[2], "72" ) == 0 ) kappa\_level = 72;

if( argc > 2 && \_stricmp( argv[2], "73" ) == 0 ) kappa\_level = 73;

if( argc > 2 && \_stricmp( argv[2], "74" ) == 0 ) kappa\_level = 74;

if( argc > 2 && \_stricmp( argv[2], "75" ) == 0 ) kappa\_level = 75;

if( argc > 2 && \_stricmp( argv[2], "76" ) == 0 ) kappa\_level = 76;

if( argc > 2 && \_stricmp( argv[2], "77" ) == 0 ) kappa\_level = 77;

if( argc > 2 && \_stricmp( argv[2], "78" ) == 0 ) kappa\_level = 78;

if( argc > 2 && \_stricmp( argv[2], "79" ) == 0 ) kappa\_level = 79;

if( argc > 2 && \_stricmp( argv[2], "80" ) == 0 ) kappa\_level = 80;

if( argc > 2 && \_stricmp( argv[2], "81" ) == 0 ) kappa\_level = 81;

if( argc > 2 && \_stricmp( argv[2], "82" ) == 0 ) kappa\_level = 82;

if( argc > 2 && \_stricmp( argv[2], "83" ) == 0 ) kappa\_level = 83;

if( argc > 2 && \_stricmp( argv[2], "84" ) == 0 ) kappa\_level = 84;

if( argc > 2 && \_stricmp( argv[2], "85" ) == 0 ) kappa\_level = 85;

if( argc > 2 && \_stricmp( argv[2], "86" ) == 0 ) kappa\_level = 86;

if( argc > 2 && \_stricmp( argv[2], "87" ) == 0 ) kappa\_level = 87;

if( argc > 2 && \_stricmp( argv[2], "88" ) == 0 ) kappa\_level = 88;

if( argc > 2 && \_stricmp( argv[2], "89" ) == 0 ) kappa\_level = 89;

if( argc > 2 && \_stricmp( argv[2], "90" ) == 0 ) kappa\_level = 90;

if( argc > 2 && \_stricmp( argv[2], "91" ) == 0 ) kappa\_level = 91;

if( argc > 2 && \_stricmp( argv[2], "92" ) == 0 ) kappa\_level = 92;

if( argc > 2 && \_stricmp( argv[2], "93" ) == 0 ) kappa\_level = 93;

if( argc > 2 && \_stricmp( argv[2], "94" ) == 0 ) kappa\_level = 94;

if( argc > 2 && \_stricmp( argv[2], "95" ) == 0 ) kappa\_level = 95;

if( argc > 2 && \_stricmp( argv[2], "96" ) == 0 ) kappa\_level = 96;

if( argc > 2 && \_stricmp( argv[2], "97" ) == 0 ) kappa\_level = 97;

if( argc > 2 && \_stricmp( argv[2], "98" ) == 0 ) kappa\_level = 98;

if( argc > 2 && \_stricmp( argv[2], "99" ) == 0 ) kappa\_level = 99;

if( argc > 2 && \_stricmp( argv[2], "100" ) == 0 ) kappa\_level = 100;

r = (float)(kappa\_level/100.0);

if( argc > 2 && \_stricmp( argv[3], "PLOT" ) == 0 ) plot = 1; else plot = 0;

if( argc > 2 && stricmp( argv[4], "NO\_PRINT" ) == 0 ) print\_long = 0; else print\_long = 1;

if( argc > 2 && \_stricmp( argv[5], "DE\_BUG" ) == 0 ) DE\_BUG = 1; else DE\_BUG = 0;

p\_value\_to\_progress = (float)0.01;

if( argc > 2 && \_stricmp( argv[6], "05" ) == 0 ) p\_value\_to\_progress = (float)0.05;

if( argc > 2 && \_stricmp( argv[6], "001" ) == 0 ) p\_value\_to\_progress = (float)0.001;

chi\_value\_to\_progress = (float)6.635;

if( argc > 2 && \_stricmp( argv[6], "05" ) == 0 ) chi\_value\_to\_progress = (float)3.841;

if( argc > 2 && \_stricmp( argv[6], "001" ) == 0 ) chi\_value\_to\_progress = (float)10.828;

marginal\_minimum = (float)10.0;

if( argc > 2 && \_stricmp( argv[7], "15" ) == 0 ) marginal\_minimum = (float)15.0;

if( argc > 2 && \_stricmp( argv[7], "20" ) == 0 ) marginal\_minimum = (float)20.0;

if( argc > 2 && \_stricmp( argv[7], "25" ) == 0 ) marginal\_minimum = (float)25.0;

if( argc > 2 && \_stricmp( argv[7], "30" ) == 0 ) marginal\_minimum = (float)30.0;

marginal\_minimum\_i = 10;

if( argc > 2 && \_stricmp( argv[7], "15" ) == 0 ) marginal\_minimum\_i = 15;

if( argc > 2 && \_stricmp( argv[7], "20" ) == 0 ) marginal\_minimum\_i = 20;

if( argc > 2 && \_stricmp( argv[7], "25" ) == 0 ) marginal\_minimum\_i = 25;

if( argc > 2 && \_stricmp( argv[7], "30" ) == 0 ) marginal\_minimum\_i = 30;

BIT64 = 0; /\*MAY WANT TO LIMIT GENETIC DATA TO 0,1,2 \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\* Set character arrays and initial key values to Zeros \*\*\*\*\* C++ does not do this automatically \*\*\*\*\*\*\*/

/\*\*\* Set character arrays NCOL\*100 = 100,000,000 = 100MB \*\*\*\*\*\*\*/

/\*\*\* Set initial key values NRUN\*NCOL\*(Num Key Variables or 9 say 10) = 15 \* 1,000,000 \* 10 = 150,000,000 = 150MB \*\*\*\*\*\*\*/

/\*\*\* Set initial memory to 100MB + 150MB = 250MB \*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (DE\_BUG == 1) printf("0000 COMMAND LINE argc = %d kappa\_level2 = %d plot3 = %d print\_long4 = %d DE\_BUG5 = %d p\_value\_to\_progress6 = %f\n",argc, kappa\_level, plot, print\_long, DE\_BUG, p\_value\_to\_progress);

if (DE\_BUG == 1) for (k =0; k <=argc; k++)printf("0011 k = %d argv[k] = %s\n",k,argv[k]);

i=j=k=l=m=n=variables=cases=letters=0;

f = oldf = 0.0;

aa=bb=cc=dd=ac=ab=bd=cd=abcd=term1=term2=term3=term4=datatemp = 0.0;

P = P1 = Eff = Sp = Se = Q = Q1 = k0\_0 = k1\_0 = 0.0;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\* read in Variable Names \*\*\* and figure out how many variables there are\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*Note, this just reads in the first line of date (names)count \t \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (DE\_BUG == 1) printf("0010 FILE OPEN Var = %d Cases = %d V\_Max = %d Cases\_max = %d\n",variables, cases, variables\_max, cases\_max);

fp = fopen(argv[1],"r"); /\*\*\* initialization and first read \*\*\*\*/

fscanf(fp,"%c",&c);

n=variables=1;

while ((c != '\n' && c != EOF)||(c != '\r' && c != EOF)) /\*\*\* Read until you hit END OF LINE (\n) or END OF FILE (EOF) \*/

{

while (c != '\t' && c != '\n' && c != '\r') /\*\*\* Read until you hit TAB or END OF LINE or <CR>(\n)\*/

{

i=i+1; /\*\*\* Number characters +1 \*\*\*\*/

fscanf(fp,"%c",&c); /\*\*\* Read another character \*\*\*/

if (c == '\n' || c == '\r') /\*\*\* End of line and last variable???\*\*\*/

{

goto dataread; /\*\*\* If so, go read some read data \*/

}

if (c == '\t') /\*\*\* Ahah a tab, so muct be a new variable \*\*\*\*/

{

variables++; /\*\*\* Up the count of variables one \*\*\*\*\*\*/

n++; /\*\*\* Worthless repetative count \*\*\*\*/

i = 0;

}

}

fscanf(fp,"%c",&c); /\*\*\* If missed above while loop, read some more \*\*\*\*/

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* simulate read in Actual Data to get counts correct\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* data[variables][n of subject]\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

dataread:

variables\_max=variables; /\*enter variables counted here as variables\_max\*/

if (DE\_BUG == 1) printf("0020 DATA READ FOR COUNTS Var = %d Cases = %d V\_Max = %d Cases\_max = %d\n",variables, cases, variables\_max, cases\_max);

n=variables=cases=1; /\* variables start as #1 not #0\*/

while (c != EOF) /\*\*\* Read numbers until EOF \*\*\*/

{

if (fscanf(fp,"%f",&f) < 0) goto dataanalysis; /\*\*\* Test for EOF \*\*\*/

fscanf(fp,"%c",&c); /\*\*\* No EOF so read away \*\*\*\*/

if (c == '\n') /\*\*\* Last data ona case so add one to that number \*\*\*/

{

cases++;

if (cases\_max < cases) cases\_max = cases;

variables=1;

goto dataread2; /\*\*\* Sneak out so you don't read past to next line \*\*\*/

}

if (c == '\t' || c == ' ') /\*\*\* One more variable \*\*\*\*/

{

variables++;

if (variables\_max < variables) variables\_max = variables; /\*\*\* Figure out how many variables\*/

}

dataread2:

n = n; /\*\*\* Dummy statement as target for goto \*\*\*\*/

}

dataanalysis:

fclose(fp);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*Dimension other arrays now that we know number of variables and cases\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*Maximize use of memory \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//float best\_value[NRUN][NCOL], best\_kappa[NRUN][NCOL], best\_variable[NRUN][NCOL], best\_chi\_square[NRUN][NCOL];

//float best\_PVP[NRUN][NCOL], best\_PVP\_denominator[NRUN][NCOL], best\_PVN[NRUN][NCOL], best\_PVN\_denominator[NRUN][NCOL], best\_LT\_GE[NRUN][NCOL];

if (DE\_BUG == 1) printf("0030 DIMENSION ARRAY Var = %d Cases = %d V\_Max = %d Cases\_max = %d\n",variables, cases, variables\_max, cases\_max);

dimension\_array\_size();

if (DE\_BUG == 1) printf("0040 ZERO VARIABLES Var = %d Cases = %d V\_Max = %d Cases\_max = %d\n",variables, cases, variables\_max, cases\_max);

zero\_variables(); /\*\*\*zero most arrays \*/

if (DE\_BUG == 1) printf("0050 ZERO ARRAYS Var = %d Cases = %d V\_Max = %d Cases\_max = %d\n",variables, cases, variables\_max, cases\_max);

for (k=0;k<NRUN+1;k++) /\*\*zero main results arrays \*/

{

for (j=1;j<variables\_max+1;j=j+1)

{

best\_value[k][j] = best\_kappa[k][j] = best\_variable[k][j] =best\_chi\_square[k][j] = 0.0;

best\_PVP[k][j]=best\_PVP\_denominator[k][j]=best\_PVN[k][j]=best\_PVN\_denominator[k][j]=best\_LT\_GE[k][j] = 0.0;

}

}

if (DE\_BUG == 1) printf("0060 ZERO NAMES ARRAYS Var = %d Cases = %d V\_Max = %d Cases\_max = %d\n",variables, cases, variables\_max, cases\_max);

for (k=0;k<variables\_max+1;k++) /\*\*\*zero var names array \*\*\*\*/

{

for (j=0;j<25;j=j+1) var[k][j] = ' ';

}

for (k=0;k<variables\_max+1;k++) /\*\*zero main data array \*/

{

for (j=0;j<cases\_max+1;j=j+1) data[k][j] = 0.0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* read in Actual Data \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* data[variables][n of subject]\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\* read in Variable Names to the var[][] array \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*Note, this just reads in the first line of date (names) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (DE\_BUG == 1) printf("0070 READ ACTUAL NAMES Var = %d Cases = %d V\_Max = %d Cases\_max = %d\n",variables, cases, variables\_max, cases\_max);

fp = fopen(argv[1],"r"); /\*\*\* initialization and first read \*\*\*\*/

fscanf(fp,"%c",&c);

n=variables=1;

while (c != '\n' && c != EOF) /\*\*\* Read until you hit END OF LINE (\n) or END OF FILE (EOF) \*/

{

while (c != '\t' && c != '\n' && c != '\r') /\*\*\* Read until you hit TAB or END OF LINE or <CR>(\n)\*/

{

i=i+1; /\*\*\* Number characters +1 \*\*\*\*/

var[n][i] = c; /\*\*\* Save that character \*\*\*\*\*/

fscanf(fp,"%c",&c); /\*\*\* Read another character \*\*\*/

if (c == '\n') /\*\*\* End of line and last variable???\*\*\*/

{

goto dataread100; /\*\*\* If so, go read some read data \*/

}

if (c == '\t') /\*\*\* Ahah a tab, so muct be a new variable \*\*\*\*/

{

variables++; /\*\*\* Up the count of variables one \*\*\*\*\*\*/

n++; /\*\*\* Worthless repetative count \*\*\*\*/

i = 0;

}

}

fscanf(fp,"%c",&c); /\*\*\* If missed above while loop, read some more \*\*\*\*/

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* actually read in Actual Data to data array \*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* data[variables][n of subject] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

dataread100:

if (DE\_BUG == 1) printf("0080 READ ACTUAL DATA Var = %d Cases = %d V\_Max = %d Cases\_max = %d\n",variables, cases, variables\_max, cases\_max);

n=variables=cases=1; /\* variables start as #1 not #0\*/

while (c != EOF) /\*\*\* Read numbers until EOF \*\*\*/

{

if (fscanf(fp,"%f",&f) < 0) goto dataanalysis200; /\*\*\* Test for EOF \*\*\*/

fscanf(fp,"%c",&c); /\*\*\* No EOF so read away \*\*\*\*/

data[variables][cases] = f;

if (c == '\n')

{

cases++;

variables=1;

goto dataread200; /\*\*\* Sneak out so you don't read past to next line \*\*\*/

}

if (c == '\t') /\*\*\* One more variable \*\*\*\*/

{

variables++;

}

dataread200:

n = n; /\*\*\* Dummy statement as target for goto \*\*\*\*/

}

dataanalysis200:

fclose(fp);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*make a copy of the main data arrays\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for (k=0;k<(variables\_max+1);k++)

{

for (j=0;j<cases\_max+1;j=j+1)

{

data2[k][j] = data[k][j];

/\*if (k > 0)printf("PASS02 k = %d j = %d data[k][j]= %f\n",k,j,data[k][j]);\*/

}

}

cases\_max = cases\_max -1; /\*\*\*\*\*Adjust for last read of EOF \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Analyze Data (MAIN LOOP) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (DE\_BUG == 1) printf("0090 BEGIN DATA ANALYSIS Var = %d Cases = %d V\_Max = %d Cases\_max = %d\n",variables, cases, variables\_max, cases\_max);

computation(1); /\*Number 1 time through\*/

printf("Computation 1 over "); print\_long\_line();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// 0.10 0.05 0.025 0.01 0.001

// 2.706 3.841 5.024 6.635 10.828

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//if (best\_chi\_square[1][1] >= 6.64 || (best\_chi\_square[1][1] >= 3.84 && p\_value\_to\_progress == 0.01)) /\*Number 2&3 times through if Chi Square < .01\*/

/\*if (best\_chi\_square[1][1] >= 10.828 || (best\_chi\_square[1][1] >= 6.635 && p\_value\_to\_progress < 0.05) || (best\_chi\_square[1][1] >= 3.841 && p\_value\_to\_progress == 0.05))\*/ /\*Number 2&3 times through if Chi Square < criteria .05/.01/.01\*/

if (best\_chi\_square[1][1] >= chi\_value\_to\_progress) /\*Number 2&3 times through if Chi Square > criteria .05/.01/.001\*/

{

printf("\*\* DO further computation beyond #1 because chi square = %8.3f p < %4.2f\n",best\_chi\_square[1][1], p\_value\_to\_progress);

separators(); /\*a bunch of lines \*/

zero\_variables(); /\*zero relevant variables \*/

recover\_data(); /\*get data as originally read in \*/

if (best\_LT\_GE[1][1] == 1.0)remove\_LT\_Data(1);

if (best\_LT\_GE[1][1] == 0.0)remove\_GE\_Data(1);

computation(2); /\*Compute and output stats for values GE cut point\*/

zero\_variables(); /\*zero relevant variables \*/

recover\_data(); /\*get data as originally read in \*/

if (best\_LT\_GE[1][1] == 1.0)remove\_GE\_Data(1);

if (best\_LT\_GE[1][1] == 0.0)remove\_LT\_Data(1);

computation(3); /\*Compute and output stats for values LT cut point\*/

}

else /\*No iteration as Chi Square too small\*/

{

printf("\*\* NO further computation because chi square %8.3f too small in computation 3\n",best\_chi\_square[1][1]);

separators();

}

printf("Computation 2&3 over "); print\_long\_line();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//if (best\_chi\_square[2][1] >= 6.64 || (best\_chi\_square[2][1] >= 3.84 && p\_value\_to\_progress > 0.01)) /\*Number 4&5 times through if Chi Square < .01\*/

//if (best\_chi\_square[2][1] >= 10.828 || (best\_chi\_square[2][1] >= 6.635 && p\_value\_to\_progress < 0.05) || (best\_chi\_square[2][1] >= 3.841 && p\_value\_to\_progress == 0.05)) Number 2&3 times through if Chi Square < criteria .05/.01/.01\*/

if (best\_chi\_square[2][1] >= chi\_value\_to\_progress) /\*Number 2&3 times through if Chi Square > criteria .05/.01/.001\*/

{

printf("\*\* DO further computation beyond #2 because chi square = %8.3f p < %4.2f\n",best\_chi\_square[2][1], p\_value\_to\_progress);

separators(); /\*a bunch of lines \*/

zero\_variables(); /\*zero relevant variables \*/

recover\_data(); /\*get data as originally read in \*/

if (best\_LT\_GE[1][1] == 1.0)remove\_LT\_Data(1);

if (best\_LT\_GE[1][1] == 0.0)remove\_GE\_Data(1);

if (best\_LT\_GE[2][1] == 1.0)remove\_LT\_Data(2);

if (best\_LT\_GE[2][1] == 0.0)remove\_GE\_Data(2);

computation(4); /\*Compute and output stats for values GE cut point\*/

zero\_variables(); /\*zero relevant variables \*/

recover\_data(); /\*get data as originally read in \*/

if (best\_LT\_GE[1][1] == 1.0)remove\_LT\_Data(1);

if (best\_LT\_GE[1][1] == 0.0)remove\_GE\_Data(1);

if (best\_LT\_GE[2][1] == 1.0)remove\_GE\_Data(2);

if (best\_LT\_GE[2][1] == 0.0)remove\_LT\_Data(2);

computation(5); /\*Compute and output stats for values LT cut point\*/

}

else /\*No iteration as Chi Square too small\*/

{

printf("\*\* NO further computation because chi square %8.3f too small in computation 2\n",best\_chi\_square[2][1]);

separators();

}

printf("Computation 4&5 over "); print\_long\_line();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//if (best\_chi\_square[3][1] >= 6.64 || (best\_chi\_square[3][1] >= 3.84 && p\_value\_to\_progress > 0.01)) /\*Number 6&7 times through if Chi Square < .01\*/

//if (best\_chi\_square[3][1] >= 10.828 || (best\_chi\_square[3][1] >= 6.635 && p\_value\_to\_progress < 0.05) || (best\_chi\_square[3][1] >= 3.841 && p\_value\_to\_progress == 0.05)) /\*Number 2&3 times through if Chi Square < criteria .05/.01/.01\*/

if (DE\_BUG == 1) printf ("BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB best\_chi\_square[3][1] = %7.3f chi\_value\_to\_progress = %7.3f\n", best\_chi\_square[3][1], chi\_value\_to\_progress); /\*Number 2&3 times through if Chi Square > criteria .05/.01/.001\*/

if (DE\_BUG == 1) printf ("BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB best\_chi\_square[3][1] = %7.3f chi\_value\_to\_progress = %7.3f\n", best\_chi\_square[3][1], chi\_value\_to\_progress); /\*Number 2&3 times through if Chi Square > criteria .05/.01/.001\*/

if (best\_chi\_square[3][1] >= chi\_value\_to\_progress) /\*Number 2&3 times through if Chi Square > criteria .05/.01/.001\*/

{

printf("\*\* DO further computation beyond #3 because chi square = %8.3f p < %4.2f\n",best\_chi\_square[3][1], p\_value\_to\_progress);

separators(); /\*a bunch of lines \*/

zero\_variables(); /\*zero relevant variables \*/

recover\_data(); /\*get data as originally read in \*/

if (best\_LT\_GE[1][1] == 1.0)remove\_GE\_Data(1);

if (best\_LT\_GE[1][1] == 0.0)remove\_LT\_Data(1);

if (best\_LT\_GE[3][1] == 1.0)remove\_LT\_Data(3);

if (best\_LT\_GE[3][1] == 0.0)remove\_GE\_Data(3);

computation(6); /\*Compute and output stats for values GE cut point\*/

zero\_variables(); /\*zero relevant variables \*/

recover\_data(); /\*get data as originally read in \*/

if (best\_LT\_GE[1][1] == 1.0)remove\_GE\_Data(1);

if (best\_LT\_GE[1][1] == 0.0)remove\_LT\_Data(1);

if (best\_LT\_GE[3][1] == 1.0)remove\_GE\_Data(3);

if (best\_LT\_GE[3][1] == 0.0)remove\_LT\_Data(3);

computation(7); /\*Compute and output stats for values LT cut point\*/

}

else /\*No iteration as Chi Square too small\*/

{

printf("\*\* NO further computation because chi square %8.3f too small in computation 3\n",best\_chi\_square[3][1]);

separators();

}

printf("Computation 6&7 over "); print\_long\_line();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//if (best\_chi\_square[4][1] >= 6.64 || (best\_chi\_square[4][1] >= 3.84 && p\_value\_to\_progress > 0.01)) /\*Number 6&7 times through if Chi Square < .01\*/

//if (best\_chi\_square[4][1] >= 10.828 || (best\_chi\_square[4][1] >= 6.635 && p\_value\_to\_progress < 0.05) || (best\_chi\_square[4][1] >= 3.841 && p\_value\_to\_progress == 0.05)) /\*Number 2&3 times through if Chi Square < criteria .05/.01/.01\*/

if (DE\_BUG == 1) printf ("BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB best\_chi\_square[4][1] = %7.3f chi\_value\_to\_progress = %7.3f\n", best\_chi\_square[4][1], chi\_value\_to\_progress); /\*Number 2&3 times through if Chi Square > criteria .05/.01/.001\*/

if (DE\_BUG == 1) printf ("BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB best\_chi\_square[4][1] = %7.3f chi\_value\_to\_progress = %7.3f\n", best\_chi\_square[4][1], chi\_value\_to\_progress); /\*Number 2&3 times through if Chi Square > criteria .05/.01/.001\*/

if (best\_chi\_square[4][1] >= chi\_value\_to\_progress) /\*Number 2&3 times through if Chi Square > criteria .05/.01/.001\*/

{

printf("\*\* DO further computation beyond #4 because chi square = %8.3f p < %4.2f\n",best\_chi\_square[4][1], p\_value\_to\_progress);

separators(); /\*a bunch of lines \*/

zero\_variables(); /\*zero relevant variables \*/

recover\_data(); /\*get data as originally read in \*/

if (best\_LT\_GE[1][1] == 1.0)remove\_LT\_Data(1);

if (best\_LT\_GE[1][1] == 0.0)remove\_GE\_Data(1);

if (best\_LT\_GE[2][1] == 1.0)remove\_LT\_Data(2);

if (best\_LT\_GE[2][1] == 0.0)remove\_GE\_Data(2);

if (best\_LT\_GE[4][1] == 1.0)remove\_LT\_Data(4);

if (best\_LT\_GE[4][1] == 0.0)remove\_GE\_Data(4);

computation(8); /\*Compute and output stats for values GE cut point\*/

zero\_variables(); /\*zero relevant variables \*/

recover\_data(); /\*get data as originally read in \*/

if (best\_LT\_GE[1][1] == 1.0)remove\_LT\_Data(1);

if (best\_LT\_GE[1][1] == 0.0)remove\_GE\_Data(1);

if (best\_LT\_GE[2][1] == 1.0)remove\_LT\_Data(2);

if (best\_LT\_GE[2][1] == 0.0)remove\_GE\_Data(2);

if (best\_LT\_GE[4][1] == 1.0)remove\_GE\_Data(4);

if (best\_LT\_GE[4][1] == 0.0)remove\_LT\_Data(4);

computation(9); /\*Compute and output stats for values LT cut point\*/

}

else /\*No iteration as Chi Square too small\*/

{

printf("\*\* NO further computation because chi square %8.3f too small in computation 4\n",best\_chi\_square[4][1]);

separators();

}

printf("Computation 8&9 over "); print\_long\_line();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//if (best\_chi\_square[5][1] >= 6.64 || (best\_chi\_square[5][1] >= 3.84 && p\_value\_to\_progress > 0.01)) /\*Number 6&7 times through if Chi Square < .01\*/

//if (best\_chi\_square[5][1] >= 10.828 || (best\_chi\_square[5][1] >= 6.635 && p\_value\_to\_progress < 0.05) || (best\_chi\_square[5][1] >= 3.841 && p\_value\_to\_progress == 0.05)) /\*Number 2&3 times through if Chi Square < criteria .05/.01/.01\*/

if (DE\_BUG == 1) printf ("BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB best\_chi\_square[5][1] = %7.3f chi\_value\_to\_progress = %7.3f\n", best\_chi\_square[5][1], chi\_value\_to\_progress); /\*Number 2&3 times through if Chi Square > criteria .05/.01/.001\*/

if (DE\_BUG == 1) printf ("BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB best\_chi\_square[5][1] = %7.3f chi\_value\_to\_progress = %7.3f\n", best\_chi\_square[5][1], chi\_value\_to\_progress); /\*Number 2&3 times through if Chi Square > criteria .05/.01/.001\*/

if (best\_chi\_square[5][1] >= chi\_value\_to\_progress) /\*Number 2&3 times through if Chi Square > criteria .05/.01/.001\*/

{

printf("\*\* DO further computation beyond #5 because chi square = %8.3f p < %4.2f\n",best\_chi\_square[5][1], p\_value\_to\_progress);

separators(); /\*a bunch of lines \*/

zero\_variables(); /\*zero relevant variables \*/

recover\_data(); /\*get data as originally read in \*/

if (best\_LT\_GE[1][1] == 1.0)remove\_LT\_Data(1);

if (best\_LT\_GE[1][1] == 0.0)remove\_GE\_Data(1);

if (best\_LT\_GE[2][1] == 1.0)remove\_GE\_Data(2);

if (best\_LT\_GE[2][1] == 0.0)remove\_LT\_Data(2);

if (best\_LT\_GE[5][1] == 1.0)remove\_LT\_Data(5);

if (best\_LT\_GE[5][1] == 0.0)remove\_GE\_Data(5);

computation(10); /\*Compute and output stats for values GE cut point\*/

zero\_variables(); /\*zero relevant variables \*/

recover\_data(); /\*get data as originally read in \*/

if (best\_LT\_GE[1][1] == 1.0)remove\_LT\_Data(1);

if (best\_LT\_GE[1][1] == 0.0)remove\_GE\_Data(1);

if (best\_LT\_GE[2][1] == 1.0)remove\_GE\_Data(2);

if (best\_LT\_GE[2][1] == 0.0)remove\_LT\_Data(2);

if (best\_LT\_GE[5][1] == 1.0)remove\_GE\_Data(5);

if (best\_LT\_GE[5][1] == 0.0)remove\_LT\_Data(5);

computation(11); /\*Compute and output stats for values LT cut point\*/

}

else /\*No iteration as Chi Square too small\*/

{

printf("\*\* NO further computation because chi square %8.3f too small in computation 5\n",best\_chi\_square[5][1]);

separators();

}

printf("Computation 10&11 over "); print\_long\_line();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//if (best\_chi\_square[6][1] >= 6.64 || (best\_chi\_square[6][1] >= 3.84 && p\_value\_to\_progress > 0.01)) /\*Number 6&7 times through if Chi Square < .01\*/

//if (best\_chi\_square[6][1] >= 10.828 || (best\_chi\_square[6][1] >= 6.635 && p\_value\_to\_progress < 0.05) || (best\_chi\_square[6][1] >= 3.841 && p\_value\_to\_progress == 0.05)) /\*Number 2&3 times through if Chi Square < criteria .05/.01/.01\*/

if (DE\_BUG == 1) printf ("BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB best\_chi\_square[6][1] = %7.3f chi\_value\_to\_progress = %7.3f\n", best\_chi\_square[6][1], chi\_value\_to\_progress); /\*Number 2&3 times through if Chi Square > criteria .05/.01/.001\*/

if (DE\_BUG == 1) printf ("BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB best\_chi\_square[6][1] = %7.3f chi\_value\_to\_progress = %7.3f\n", best\_chi\_square[6][1], chi\_value\_to\_progress); /\*Number 2&3 times through if Chi Square > criteria .05/.01/.001\*/

if (best\_chi\_square[6][1] >= chi\_value\_to\_progress) /\*Number 2&3 times through if Chi Square > criteria .05/.01/.001\*/

{

printf("\*\* DO further computation beyond #6 because chi square = %8.3f p < %4.2f\n",best\_chi\_square[6][1], p\_value\_to\_progress);

separators(); /\*a bunch of lines \*/

zero\_variables(); /\*zero relevant variables \*/

recover\_data(); /\*get data as originally read in \*/

if (best\_LT\_GE[1][1] == 1.0)remove\_GE\_Data(1);

if (best\_LT\_GE[1][1] == 0.0)remove\_LT\_Data(1);

if (best\_LT\_GE[3][1] == 1.0)remove\_LT\_Data(3);

if (best\_LT\_GE[3][1] == 0.0)remove\_GE\_Data(3);

if (best\_LT\_GE[6][1] == 1.0)remove\_LT\_Data(6);

if (best\_LT\_GE[6][1] == 0.0)remove\_GE\_Data(6);

computation(12); /\*Compute and output stats for values GE cut point\*/

zero\_variables(); /\*zero relevant variables \*/

recover\_data(); /\*get data as originally read in \*/

if (best\_LT\_GE[1][1] == 1.0)remove\_GE\_Data(1);

if (best\_LT\_GE[1][1] == 0.0)remove\_LT\_Data(1);

if (best\_LT\_GE[3][1] == 1.0)remove\_LT\_Data(3);

if (best\_LT\_GE[3][1] == 0.0)remove\_GE\_Data(3);

if (best\_LT\_GE[6][1] == 1.0)remove\_GE\_Data(6);

if (best\_LT\_GE[6][1] == 0.0)remove\_LT\_Data(6);

computation(13); /\*Compute and output stats for values LT cut point\*/

}

else /\*No iteration as Chi Square too small\*/

{

printf("\*\* NO further computation because chi square %8.3f too small in computation 6\n",best\_chi\_square[6][1]);

separators();

}

printf("Computation 12&13 over "); print\_long\_line();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//if (best\_chi\_square[7][1] >= 6.64 || (best\_chi\_square[7][1] >= 3.84 && p\_value\_to\_progress > 0.01)) /\*Number 6&7 times through if Chi Square < .01\*/

//if (best\_chi\_square[7][1] >= 10.828 || (best\_chi\_square[7][1] >= 6.635 && p\_value\_to\_progress < 0.05) || (best\_chi\_square[7][1] >= 3.841 && p\_value\_to\_progress == 0.05)) /\*Number 2&3 times through if Chi Square < criteria .05/.01/.01\*/

if (DE\_BUG == 1) printf ("BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB best\_chi\_square[7][1] = %7.3f chi\_value\_to\_progress = %7.3f\n", best\_chi\_square[7][1], chi\_value\_to\_progress); /\*Number 2&3 times through if Chi Square > criteria .05/.01/.001\*/

if (DE\_BUG == 1) printf ("BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB best\_chi\_square[7][1] = %7.3f chi\_value\_to\_progress = %7.3f\n", best\_chi\_square[7][1], chi\_value\_to\_progress); /\*Number 2&3 times through if Chi Square > criteria .05/.01/.001\*/

if (best\_chi\_square[7][1] >= chi\_value\_to\_progress) /\*Number 2&3 times through if Chi Square > criteria .05/.01/.001\*/

{

printf("\*\* DO further computation beyond #7 because chi square = %8.3f p < %4.2f\n",best\_chi\_square[7][1], p\_value\_to\_progress);

separators(); /\*a bunch of lines \*/

zero\_variables(); /\*zero relevant variables \*/

recover\_data(); /\*get data as originally read in \*/

if (best\_LT\_GE[1][1] == 1.0)remove\_GE\_Data(1);

if (best\_LT\_GE[1][1] == 0.0)remove\_LT\_Data(1);

if (best\_LT\_GE[3][1] == 1.0)remove\_GE\_Data(3);

if (best\_LT\_GE[3][1] == 0.0)remove\_LT\_Data(3);

if (best\_LT\_GE[7][1] == 1.0)remove\_LT\_Data(7);

if (best\_LT\_GE[7][1] == 0.0)remove\_GE\_Data(7);

computation(14); /\*Compute and output stats for values GE cut point\*/

zero\_variables(); /\*zero relevant variables \*/

recover\_data(); /\*get data as originally read in \*/

if (best\_LT\_GE[1][1] == 1.0)remove\_GE\_Data(1);

if (best\_LT\_GE[1][1] == 0.0)remove\_LT\_Data(1);

if (best\_LT\_GE[3][1] == 1.0)remove\_GE\_Data(3);

if (best\_LT\_GE[3][1] == 0.0)remove\_LT\_Data(3);

if (best\_LT\_GE[7][1] == 1.0)remove\_GE\_Data(7);

if (best\_LT\_GE[7][1] == 0.0)remove\_LT\_Data(7);

computation(15); /\*Compute and output stats for values LT cut point\*/

}

else /\*No iteration as Chi Square too small\*/

{

printf("\*\* NO further computation because chi square %8.3f too small in computation 7\n",best\_chi\_square[7][1]);

separators();

}

printf("Computation 14&15 over "); print\_long\_line();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

print\_summary();

//if (best\_chi\_square[1][1] >= 6.64 || (best\_chi\_square[1][1] >= 3.84 && p\_value\_to\_progress > 0.01)) print\_diagram();

//if (best\_chi\_square[1][1] >= 10.828 || (best\_chi\_square[1][1] >= 6.635 && p\_value\_to\_progress < 0.05) || (best\_chi\_square[1][1] >= 3.841 && p\_value\_to\_progress == 0.05)) print\_diagram(); /\*Number 2&3 times through if Chi Square < criteria .05/.01/.01\*/

if (best\_chi\_square[1][1] >= chi\_value\_to\_progress) print\_diagram();

else

{

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* No cut points are significant at p < %4.2f \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n", p\_value\_to\_progress);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* No cut points are significant at p < %4.2f \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n", p\_value\_to\_progress);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* No cut points are significant at p < %4.2f \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n", p\_value\_to\_progress);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* No cut points are significant at p < %4.2f \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n", p\_value\_to\_progress);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* No cut points are significant at p < %4.2f \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n", p\_value\_to\_progress);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* No cut points are significant at p < %4.2f \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n", p\_value\_to\_progress);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* No cut points are significant at p < %4.2f \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n", p\_value\_to\_progress);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* No cut points are significant at p < %4.2f \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n", p\_value\_to\_progress);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* No cut points are significant at p < %4.2f \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n", p\_value\_to\_progress);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* No cut points are significant at p < %4.2f \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n", p\_value\_to\_progress);

}

return 0;

} /\*\*\*\*\*\*end of Main() \*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function Definitions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Computation Function \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void computation(int run) /\*New type function definition\*/

{

printf("\*\*\*\*\*\*\*\*\* Computation Number %d\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n", run);

printf("\*\*\*\*\*\*\*\*\* Computation Number %d\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n", run);

printf("\*\*\*\*\*\*\*\*\* Computation Number %d\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n", run);

k2 = 0;

j2 = 0;

//printf ("00000000000000 %6d %6d %6d %6.2f %6.2f \n", k, j, cases\_max, data[k][j], variables\_minimum[k]);

for (k=1;k<=variables\_max;k++) /\*\*\* Now it gets easy \*\*\*/

{

}

//printf ("00000000000001 %6d %6d %6d %6.2f %6.2f \n", k, j, cases\_max, data[k][j], variables\_minimum[k]);

for (k=1;k<=variables\_max;k++) /\*\*\* Now it gets easy \*\*\*/

{

//printf ("00000000000002 %6d %6d %6d %6.2f %6.2f \n", k, j, cases\_max, data[k][j], variables\_minimum[k]);

for (j=1;j<=cases\_max;j++)

{

//if ((data[k][j] != MISSING\_INT)&&(data[k][j] != MISSING\_FLOAT)) /\*\*\*\*missing values = -9999.99\*\*\*\*/

if (data[k][j] != MISSING\_INT) /\*\*\*\*missing values = -9999\*\*\*\*/

{

variables\_sum[k] = variables\_sum[k] + data[k][j];

variables\_n[k]=variables\_n[k] + 1.0;

variables\_mean[k] = variables\_sum[k]/variables\_n[k]; /\*\*\*\*\* get means for variables \*\*\*\*\*\*\*/

temp\_data\_ss = data[k][j];

if (temp\_data\_ss < 0) temp\_data\_ss = temp\_data\_ss \* -1.0;

variables\_sumrt[k]=variables\_sumrt[k] + temp\_data\_ss; /\*\*sum \*\*/

variables\_sumsq[k]=variables\_sumsq[k]+(data[k][j]\*data[k][j]); /\*\*sum of squares \*\*/

if (data[k][j] > variables\_maximum[k]) variables\_maximum[k] = data[k][j];

if (data[k][j] < variables\_minimum[k]) variables\_minimum[k] = data[k][j];

/\*float sdndev(float sumsq,float sumrt,float n)\*/

//printf ("00000000000001 %6d \t %6d \t %6d \t %6.2f \t %6.2f \n", k, j, cases\_max, data[k][j], variables\_minimum[k]);

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* True Positives Etc. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for (k=1;k<=variables\_max-1;k++) /\*\*\*\*\*\*for every variable \*\*\*\*\*\*\*\*/

{

for (j=1;j<=cases\_max;j++) /\*\*\*\*\*\*for every value of that variable \*\*\*\*\*\*\*\*/

{

for (l=1;l<=cases\_max;l++) /\*\*\*\*\*\*calculate if that value generated a good selection\*\*\*\*\*\*/

{

//if (((data[k][j] != MISSING\_INT)&&(data[k][j] != MISSING\_FLOAT)) && ((data[k][l] != MISSING\_INT)||(data[k][l] != MISSING\_FLOAT)))

if (data[k][j] != MISSING\_INT && data[k][l] != MISSING\_INT)

{

if (data[k][l] >= data[k][j] && data[variables\_max][l] == 1.0) True\_Positives[k][j]++; /\*\* a sum of True Positives\*\*\*/

if (data[k][l] >= data[k][j] && data[variables\_max][l] == 0.0) False\_Positives[k][j]++; /\*\* b sum of False Positives\*\*\*/

if (data[k][l] < data[k][j] && data[variables\_max][l] == 1.0) False\_Negatives[k][j]++; /\*\* c sum of False Negative \*\*\*/

if (data[k][l] < data[k][j] && data[variables\_max][l] == 0.0) True\_Negatives [k][j]++; /\*\* d sum of True Negative \*\*\*/

}

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Output of Mean Results \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (print\_long == 1) printf("\*\* Var\tVariables \t\t\t\t n\tMean\tMin\tMax\tSD\n");

for (k=1;k<=variables\_max;k++)

{

if (print\_long == 1) printf("\*\* %d\t",k);

if (print\_long == 1) for (j=1;j<25;j++)

{

printf("%c",var[k][j]);

}

if (print\_long == 1) printf(" "); /\*Hopefully this adds 25 spaces to reflect cuting down var length from 100->50->25. \*/

if (k != variables\_max) if (print\_long == 1) printf("\t\t%6.0f\t%7.2f\t%7.2f\t%7.2f\t%7.2f",

variables\_n[k],variables\_mean[k],variables\_minimum[k],variables\_maximum[k], sdndev(variables\_sumsq[k], variables\_sumrt[k], variables\_n[k]));

if (k != variables\_max) if (print\_long == 1) printf("\n");

}

if (print\_long == 1) printf("\t\t%6.0f\t%7.2f\t%7.2f\t%7.2f",

variables\_n[variables\_max],variables\_mean[variables\_max],variables\_minimum[variables\_max],variables\_maximum[variables\_max]);

if (print\_long == 1) printf("\tGold Standard with Proportion Positive = %5.2f\n", variables\_mean[variables\_max]);

if (run == 1)

{

total\_n = (int)variables\_n[variables\_max];

percent\_positive = (100\*variables\_mean[variables\_max]);

percent\_positivef = (100\*variables\_mean[variables\_max]);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Sort Results in ascending order for each variable \*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (print\_long == 1) {

for (k=1;k<=variables\_max-1;k++)

{

startj:

for (j=1;j<=cases\_max-1;j++)

{

if(data[k][j]>data[k][j+1])

{

datatemp = data[k][j];

data[k][j] = data[k][j+1];

data[k][j+1] = datatemp;

datatemp = data[variables\_max][j];

data[variables\_max][j] = data[variables\_max][j+1];

data[variables\_max][j+1] = datatemp;

datatemp = True\_Positives[k][j];

True\_Positives[k][j] = True\_Positives[k][j+1];

True\_Positives[k][j+1] = datatemp;

datatemp = True\_Negatives[k][j];

True\_Negatives[k][j] = True\_Negatives[k][j+1];

True\_Negatives[k][j+1] = datatemp;

datatemp = False\_Positives[k][j];

False\_Positives[k][j] = False\_Positives[k][j+1];

False\_Positives[k][j+1] = datatemp;

datatemp = False\_Negatives[k][j];

False\_Negatives[k][j] = False\_Negatives[k][j+1];

False\_Negatives[k][j+1] = datatemp;

datatemp = chi\_square[k][j];

chi\_square[k][j] = chi\_square[k][j+1];

chi\_square[k][j+1] = datatemp;

datatemp = sensitivity[k][j];

sensitivity[k][j] = sensitivity[k][j+1];

sensitivity[k][j+1] = datatemp;

datatemp = specificity[k][j];

specificity[k][j] = specificity[k][j+1];

specificity[k][j+1] = datatemp;

goto startj;

}

}

}

} /\*End if (print\_long == 1) \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Print Results in ascending order for each variable value \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (print\_long == 1) printf ("\*\*\*\*\*\*\*\*\*\* Results in ascending order for each variable value (Mg<10 means marginal count < 10 and cannot compute valid significance for Chi Square)\n");

if (print\_long == 1) printf ("\*\* Var #\tn\tCriter\tValue\tP\tQ\tsens\tspec\teffic\tk0\_0\tk0\_%2d\tk1\_0\tPVP\tnum\tden\tPVN\tnum\tden\tChi\_Sq\tSig\n",kappa\_level);

for (k=1;k<=variables\_max-1;k++) /\*\*\*\*\*\*for every variable \*\*\*\*\*\*\*\*/

{

if (print\_long == 1) printf("\*\* Variable = %4d ",k);

for (j=0;j<25 && var[k][j] != (int)' ';j++)

{

if (print\_long == 1) printf("%c",var[k][j]);

}

if (print\_long == 1) printf(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n",k);

for (j=1;j<=cases\_max;j++) /\*\*for every value of that variable \*\*print that value generated a good selection proportion\*\*\*\*\*\*/

{

//if ((data[k][j] != data[k][j+1] || j==cases\_max) && ((data[k][j] != MISSING\_INT)&&(data[k][j] != MISSING\_FLOAT))) /\*\*\*leave out duplicate results.. save a tree\*\*\*/

if ((data[k][j] != data[k][j+1] || j==cases\_max) && (data[k][j] != MISSING\_INT)) /\*\*\*leave out duplicate results.. save a tree\*\*\*/

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*Formulas from HC Kraemer, Evaluating Medical Tests, p. x \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*Sage Publications, Newbury Park, Ca 1992 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*Need below for switch \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\* a sum of True Positives\*\*\*/

/\*\* b sum of False Positives\*\*\*/

/\*\* c sum of False Negative \*\*\*/

/\*\* d sum of True Negative \*\*\*/

aa=True\_Positives[k][j]; /\*\*predicted 1 actual 1\*\*/

bb=False\_Positives[k][j]; /\*\*predicted 1 actual 0\*\*/

cc=False\_Negatives[k][j]; /\*\*predicted 0 actual 1\*\*/

dd=True\_Negatives[k][j]; /\*\*predicted 0 actual 0\*\*/

Se = aa/(aa + bb); /\*\*\*sensitivity\*\*\*/

Sp = dd/(cc + dd); /\*\*\*specificity\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

negative\_switch = 0; /\*\*\*\*\*\*\*\*\*\*\*\* start as positive correlation \*/

if ((Se+Sp) < 1 && plot == 0) /\*\*\*\*\*\*\*\*\*\*\*\* MAJOR SWITCH based on negative vs postive correlation of variable with gold standard \*/

/\*\*\*\*\*\*\*\*\*\*\*\* OR just data for plotting \*/

/\*\*\*\*\*\*\*\*\*\*\*\* which are always printed as GE to get smooth ROC curves \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

{

negative\_switch = 1; /\*\*\*\*\*\*\*\*\*\*\*\* change to negative correlation \*/

datatemp = aa;

aa = bb;

bb = datatemp;

datatemp = cc;

cc = dd;

dd = datatemp;

}/\*End of if Se+sp < 1\*/

aa=True\_Positives[k][j]; /\*\*predicted 1 actual 1\*\*/

bb=False\_Positives[k][j]; /\*\*predicted 1 actual 0\*\*/

cc=False\_Negatives[k][j]; /\*\*predicted 0 actual 1\*\*/

dd=True\_Negatives[k][j]; /\*\*predicted 0 actual 0\*\*/

ac=aa+cc; /\*\* actual 1 actual 0 \*\*/

ab=aa+bb; /\*\* predicted 1 aa bb ab \*\*/

bd=bb+dd; /\*\* predicted 0 cc dd cd \*\*/

cd=cc+dd; /\*\* marginal counts ac bd abcd \*\*/

abcd=aa+bb+cc+dd;

TP = aa/abcd;

FP = bb/abcd;

FN = cc/abcd;

TN = dd/abcd;

term1 = (aa-(ac\*ab/abcd)) \* (aa-(ac\*ab/abcd)) / (ac\*ab/abcd);

term2 = (bb-(bd\*ab/abcd)) \* (bb-(bd\*ab/abcd)) / (bd\*ab/abcd);

term3 = (cc-(ac\*cd/abcd)) \* (cc-(ac\*cd/abcd)) / (ac\*cd/abcd);

term4 = (dd-(bd\*cd/abcd)) \* (dd-(bd\*cd/abcd)) / (bd\*cd/abcd);

/\*\*\*termn = (cell value - expected cell value)squared / expected cell value \*\*\*/

chi\_square[k][j]= term1 + term2 + term3 + term4 ;

if (ac == 0 || ab == 0 || bd == 0 || cd == 0) chi\_square[k][j] = -9.99; /\*\* divide by zero no no \*/

if (ac < marginal\_minimum || ab < marginal\_minimum || bd < marginal\_minimum || cd < marginal\_minimum) chi\_square[k][j] = -9.9; /\*\* cells too small to compute chi square \*/

P = TP+FN; /\*\*\* Prevalence - fixed \*\*\*/

P1 = 1.0-P;

Q = TP+FP; /\*\*\* Level of test- varies with test \*\*\*/

if (negative\_switch == 1) Q = (1.0-Q);

Q1 = 1.0-Q;

Se = TP/P; /\*\*\*sensitivity\*\*\*/

if (negative\_switch == 1) Se = (1.0-Se);

sensitivity[k][j]=Se;

Sp = TN/P1;

if (negative\_switch == 1) Sp = (1.0-Sp);

specificity[k][j]=Sp;

/\*\*\*specificity\*\*\*/

PVP[k][j] = aa/ab;

PVP\_denominator[k][j] = ab;

if (ab == 0.0) PVP[k][j]= -9.99;

/\*\*\*\*\*PVP = TP/Q; cannot get to work \*\*\*\*\*/

PVN[k][j] = dd/cd;

PVN\_denominator[k][j] = cd;

if (cd == 0.0) PVN[k][j]= -9.99;

/\*PVN = TN/Q1; cannot get to work \*\*\*\*\*/

if (Q == 0) PVP[k][j] = -9.99; /\*\* identify division by zero cases \*\*/

if (Q1 == 0) PVN[k][j] = -9.99; /\*\* identify division by zero cases \*\*/

Eff = TP+TN; /\*\*\*efficiency\*\*\*/

if (negative\_switch == 1) Eff = (1.0-Eff);

k1\_0 = (Se-Q)/Q1;

if (Q1 == 0) k1\_0 = -9.99;

k0\_0 = (Sp-Q1)/Q;

if (Q == 0) k0\_0 = -9.99;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*K(r) = (ad-bc)/PQ'r+P'Qr' will replace k0\_5 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*k0\_5[k][j] = (Eff - P\*Q - P1\*Q1)/(1.0 - P\*Q - P1\*Q1); \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

k0\_5[k][j] = (( (aa/abcd) \* (dd/abcd) )-( (bb/abcd) \* (cc/abcd) ))/(P\*Q1\*r + P1\*Q\*(1.0-r));

if (negative\_switch == 1)

{

k0\_5[k][j] = (( (bb/abcd) \* (cc/abcd) )-( (aa/abcd) \* (dd/abcd) ))/(P\*Q1\*r + P1\*Q\*(1.0-r));

}

if (ac < marginal\_minimum || ab < marginal\_minimum || bd < marginal\_minimum || cd < marginal\_minimum) k0\_5[k][j] = -9.9; /\*\* cells too small to compute kappa \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\* printing only below here \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (negative\_switch == 0 && ((data[k][j] != MISSING\_INT)&&(data[k][j] != MISSING\_FLOAT)))

{

LT\_GE[k][j] = 1.0;

if (1==1 /\*ab > 9 && cd > 9 && ac > 9 && bd > 9\*/)

{

if (DE\_BUG == 1) printf ("Print Number 1 Neg switch ==0\n"); /\*\*/

if (print\_long == 1) printf ("%d\t%5.0f\tGE\t%8.3f\t%5.3f\t%5.3f\t%5.3f\t%5.3f\t%5.3f\t%5.3f\t%5.3f\t%5.3f\t%4.2f\t%4.0f\t%4.0f\t%4.2f\t%5.0f\t%5.0f\t%6.2f\t"

,k,variables\_n[k],data[k][j],

P,

Q,

//Se, /\*\*\*sensitivity\*\*\*/

//Sp, /\*\*\*specificity\*\*\*/

sensitivity[k][j],

specificity[k][j],

Eff, /\*\*\*efficiency\*\*\*/

k0\_0,

k0\_5[k][j] ,

k1\_0,

PVP[k][j],

aa,ab,

PVN[k][j],

dd,cd,

chi\_square[k][j]);

if (ab >= marginal\_minimum && cd >= marginal\_minimum && ac >= marginal\_minimum && bd >= marginal\_minimum && print\_long == 1)

{

if (chi\_square[k][j] >= 3.84 && chi\_square[k][j] < 6.64) printf("\*"); /\*\*p <.05 etc.\*\*/

if (chi\_square[k][j] >= 6.64 && chi\_square[k][j] < 10.83) printf("\*\*"); /\*\*p <.01 etc.\*\*/

if (chi\_square[k][j] >= 10.83) printf("\*\*\*"); /\*\*p <.001 etc.\*\*/

}

else

{

if (chi\_square[k][j] == -9.99)

{

if (print\_long == 1) printf("Cell=0"); /\*\*division by zero\*\*/

}

else

{

if (print\_long == 1) printf("Mg<%d", marginal\_minimum\_i); /\*\*too few cells for accurate Chi Square\*\*/

}

}

if (print\_long == 1) printf("\n");

}/\*\*\*\*\*end if (ab > 9 && cd > 9 && ac > 9 && bd > 9) \*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}/\*\*\*\*\*end if negative\_switch == 0 \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (negative\_switch == 1 && ((data[k][j] != MISSING\_INT)&&(data[k][j] != MISSING\_FLOAT)))

{

LT\_GE[k][j] = 0.0;

PVP[k][j] = cc/cd; if (cd == 0) PVP[k][j] = -9.99;

PVP\_denominator[k][j] = cd;

PVN[k][j] = bb/ab; if (ab == 0) PVN[k][j] = -9.99;

PVN\_denominator[k][j] = ab;

if (1==1 /\*ab > 9 && cd > 9 && ac > 9 && bd > 9\*/)

{

if (DE\_BUG == 1) printf ("Print Number 2 Neg switch ==1\n"); /\*\*/

if (print\_long == 1) printf ("%d\t%5.0f\tLT\t%8.3f\t%5.3f\t%5.3f\t%5.3f\t%5.3f\t%5.3f\t%5.3f\t%5.3f\t%5.3f\t%4.2f\t%4.0f\t%4.0f\t%4.2f\t%5.0f\t%5.0f\t%6.2f\t"

,k,variables\_n[k],data[k][j],

P,

Q,

//Se, /\*\*\*sensitivity\*\*\*/

//Sp, /\*\*\*specificity\*\*\*/

sensitivity[k][j],

specificity[k][j],

Eff, /\*\*\*efficiency\*\*\*/

k0\_0,

k0\_5[k][j] ,

k1\_0,

PVP[k][j],

cc,cd,

PVN[k][j],

bb,ab,

chi\_square[k][j]);

if (ab >= marginal\_minimum && cd >= marginal\_minimum && ac >= marginal\_minimum && bd >= marginal\_minimum && print\_long == 1)

{

if (chi\_square[k][j] >= 3.84 && chi\_square[k][j] < 6.64) printf("\*"); /\*\*p <.05 etc.\*\*/

if (chi\_square[k][j] >= 6.64 && chi\_square[k][j] < 10.83) printf("\*\*"); /\*\*p <.01 etc.\*\*/

if (chi\_square[k][j] >= 10.83) printf("\*\*\*"); /\*\*p <.001 etc.\*\*/

}

else

{

if (chi\_square[k][j] == -9.99)

{

if (print\_long == 1) printf("Cell=0"); /\*\*division by zero\*\*/

}

else

{

if (print\_long == 1) printf("Mg<%d", marginal\_minimum\_i); /\*\*too few cells for accurate Chi Square\*\*/

}

}

if (print\_long == 1) printf("\n");

}///\*\*\*\* if (ab > 9 && cd > 9 && ac > 9 && bd > 9) \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}/\*\*\*\*\*end if negative\_switch == 1 \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}/\*End of if (data[k][j] != data[k][j+1] || j==cases\_max) /\*\*\*leave out duplicate results.. save a tree\*\*\*/

}/\*End of for (j=1;j<=cases\_max;j++) /\*\*for every value of that variable \*\*print that value generated a good selection proportion\*\*\*\*\*\*/

}/\*End of for (k=1;k<=variables\_max-1;k++) /\*\*\*\*\*\*for every variable \*\*\*\*\*\*\*\*/

/\*printf("\f"); like to put page break here \*/

if (print\_long == 1) printf("\*\*\n"); /\*like to put separators here \*/

if (print\_long == 1) if (print\_long == 1) printf("\*\*\n"); /\*like to put separators here \*/

printf("\*\*\n"); /\*like to put separators here \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Identify Best Results in descending order by k0\_5 \*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (plot == 0) /\*Do this only if not a plot data run\*/

{

for (k=1;k<=variables\_max-1;k++)

{

for (j=1;j<=cases\_max-1;j++)

{

aa=True\_Positives[k][j]; /\*\*predicted 1 actual 1\*\*/

bb=False\_Positives[k][j]; /\*\*predicted 1 actual 0\*\*/

cc=False\_Negatives[k][j]; /\*\*predicted 0 actual 1\*\*/

dd=True\_Negatives[k][j]; /\*\*predicted 0 actual 0\*\*/

ac=aa+cc; /\*\* actual 1 actual 0 \*\*/

ab=aa+bb; /\*\* predicted 1 aa bb ab \*\*/

bd=bb+dd; /\*\* predicted 0 cc dd cd \*\*/

cd=cc+dd; /\*\* marginal counts ac bd abcd \*\*/

/\*printf("&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&\n");\*/

/\*printf ("1###################### k = %5d j = %5d k0\_5[k][j] = %5.3f ab = %10.3f cd = %10.3f ac = %10.3f bd = %10.3f\n",k,j,k0\_5[k][j], ab,cd,ac,bd);\*/

//if (ab < 10 || cd < 10 || ac < 10 || bd < 10) k0\_5[k][j] = -0.99;

if (ab < marginal\_minimum || cd < marginal\_minimum || ac < marginal\_minimum || bd < marginal\_minimum) k0\_5[k][j] = -0.99; /\* change of marginal counts to new variable \*/

/\*printf ("2###################### k = %5d j = %5d k0\_5[k][j] = %5.3f ab = %10.3f cd = %10.3f ac = %10.3f bd = %10.3f\n",k,j,k0\_5[k][j], ab,cd,ac,bd);\*/

}

}

/\*print\_kappa();\*/

/\* note, the following conditionals may not be necessary \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* it may run OK with just the == 0 option, but am still researching this \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (print\_long == 1)

{

for (k=1;k<=variables\_max-1;k++)

{

//startjjj:

for (j=1;j<=cases\_max-1;j++)

{

if(k0\_5[k][j+1]>k0\_5[k][j])

{

datatemp = data[k][j];

data[k][j] = data[k][j+1];

data[k][j+1] = datatemp;

datatemp = data[variables\_max][j];

data[variables\_max][j] = data[variables\_max][j+1];

data[variables\_max][j+1] = datatemp;

datatemp = True\_Positives[k][j];

True\_Positives[k][j] = True\_Positives[k][j+1];

True\_Positives[k][j+1] = datatemp;

datatemp = True\_Negatives[k][j];

True\_Negatives[k][j] = True\_Negatives[k][j+1];

True\_Negatives[k][j+1] = datatemp;

datatemp = False\_Positives[k][j];

False\_Positives[k][j] = False\_Positives[k][j+1];

False\_Positives[k][j+1] = datatemp;

datatemp = False\_Negatives[k][j];

False\_Negatives[k][j] = False\_Negatives[k][j+1];

False\_Negatives[k][j+1] = datatemp;

datatemp = k0\_5[k][j];

k0\_5[k][j] = k0\_5[k][j+1];

k0\_5[k][j+1] = datatemp;

datatemp = chi\_square[k][j];

chi\_square[k][j] = chi\_square[k][j+1];

chi\_square[k][j+1] = datatemp;

datatemp = PVP[k][j];

PVP[k][j] = PVP[k][j+1];

PVP[k][j+1] = datatemp;

datatemp = PVP\_denominator[k][j];

PVP\_denominator[k][j] = PVP\_denominator[k][j+1];

PVP\_denominator[k][j+1] = datatemp;

datatemp = PVN[k][j];

PVN[k][j] = PVN[k][j+1];

PVN[k][j+1] = datatemp;

datatemp = sensitivity[k][j];

sensitivity[k][j] = sensitivity[k][j+1];

sensitivity[k][j+1] = datatemp;

datatemp = specificity[k][j];

specificity[k][j] = specificity[k][j+1];

specificity[k][j+1] = datatemp;

datatemp = PVN\_denominator[k][j];

PVN\_denominator[k][j] = PVN\_denominator[k][j+1];

PVN\_denominator[k][j+1] = datatemp;

datatemp = LT\_GE[k][j];

LT\_GE[k][j] = LT\_GE[k][j+1];

LT\_GE[k][j+1] = datatemp;

/\*goto startjjj;\*/ j = 0;

}

}

}

}/\*end if print\_long = 1\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (print\_long == 1)

{

for (k=1;k<=variables\_max-1;k++)

{

if (DE\_BUG == 1) printf("0100 INSIDE k COMPUTATION() V\_Max = %d k = %d\n",variables\_max, k);

for (j=1;j<=cases\_max-1;j++)

{

if (DE\_BUG == 1) printf("0101 INSIDE j Cases\_max = %d j = %d k0\_5[k][j+1] = %f k0\_5[k][1] =%f\n",variables\_max, j, k0\_5[k][j+1], k0\_5[k][1]);

if (DE\_BUG == 1) printf("0102111 INSIDE j Cases\_max = %d k = %d j = %d sensitivity[k][j+1] = %f sensitivity[k][1] =%f\n",variables\_max, k, j, sensitivity[k][j+1], sensitivity[k][1]);

if (DE\_BUG == 1) printf("0102111 INSIDE j Cases\_max = %d k = %d j = %d specificity[k][j+1] = %f specificity[k][1] =%f\n",variables\_max, k, j, specificity[k][j+1], specificity[k][1]);

if(k0\_5[k][j+1]>k0\_5[k][1])

{

if (DE\_BUG == 1) printf("0102 INSIDE k0\_5 V\_Max = %d k = %d Cases\_max = %d j = %d\n", variables\_max, k,variables\_max, j);

if (DE\_BUG == 1) printf("0102222 INSIDE j Cases\_max = %d k = %d j = %d sensitivity[k][j+1] = %f sensitivity[k][1] =%f\n",variables\_max, k, j, sensitivity[k][j+1], sensitivity[k][1]);

if (DE\_BUG == 1) printf("0102222 INSIDE j Cases\_max = %d k = %d j = %d specificity[k][j+1] = %f specificity[k][1] =%f\n",variables\_max, k, j, specificity[k][j+1], specificity[k][1]);

datatemp = data[k][1];

data[k][1] = data[k][j+1];

data[k][j+1] = datatemp;

datatemp = data[variables\_max][1];

data[variables\_max][1] = data[variables\_max][j+1];

data[variables\_max][j+1] = datatemp;

datatemp = True\_Positives[k][1];

True\_Positives[k][1] = True\_Positives[k][j+1];

True\_Positives[k][j+1] = datatemp;

datatemp = True\_Negatives[k][1];

True\_Negatives[k][1] = True\_Negatives[k][j+1];

True\_Negatives[k][j+1] = datatemp;

datatemp = False\_Positives[k][1];

False\_Positives[k][1] = False\_Positives[k][j+1];

False\_Positives[k][j+1] = datatemp;

datatemp = False\_Negatives[k][1];

False\_Negatives[k][1] = False\_Negatives[k][j+1];

False\_Negatives[k][j+1] = datatemp;

datatemp = k0\_5[k][1];

k0\_5[k][1] = k0\_5[k][j+1];

k0\_5[k][j+1] = datatemp;

datatemp = chi\_square[k][1];

chi\_square[k][1] = chi\_square[k][j+1];

chi\_square[k][j+1] = datatemp;

datatemp = PVP[k][1];

PVP[k][1] = PVP[k][j+1];

PVP[k][j+1] = datatemp;

datatemp = PVP\_denominator[k][1];

PVP\_denominator[k][1] = PVP\_denominator[k][j+1];

PVP\_denominator[k][j+1] = datatemp;

datatemp = PVN[k][1];

PVN[k][1] = PVN[k][j+1];

PVN[k][j+1] = datatemp;

datatemp = sensitivity[k][j];

sensitivity[k][j] = PVN[k][j+1];

sensitivity[k][j+1] = datatemp;

datatemp = specificity[k][j];

specificity[k][j] = PVN[k][j+1];

specificity[k][j+1] = datatemp;

datatemp = PVN\_denominator[k][1];

PVN\_denominator[k][1] = PVN\_denominator[k][j+1];

PVN\_denominator[k][j+1] = datatemp;

datatemp = LT\_GE[k][1];

LT\_GE[k][1] = LT\_GE[k][j+1];

LT\_GE[k][j+1] = datatemp;

if (DE\_BUG == 1) printf("0102223 INSIDE j Cases\_max = %d k = %d j = %d sensitivity[k][j+1] = %f sensitivity[k][1] =%f\n",variables\_max, k, j, sensitivity[k][j+1], sensitivity[k][1]);

if (DE\_BUG == 1) printf("0102223 INSIDE j Cases\_max = %d k = %d j = %d specificity[k][j+1] = %f specificity[k][1] =%f\n",variables\_max, k, j, specificity[k][j+1], specificity[k][1]);

/\*goto startjjj; j = 0; Just do this loop once and put best kappa in [j=1]\*/

}

}

}

}/\*end if print long = 1\*/

if (print\_long == 0)

{

if (DE\_BUG == 1) printf("0110 LOOP COMPLETED k0\_5[k][1] = %f\n",k0\_5[k][1]);

datatemp = 0.0;

for (k=1;k<=variables\_max-1;k++)

{

for (j=1;j<=cases\_max-1;j++)

{

if (DE\_BUG == 1) printf("0111 ---------- k = %d j = %d k0\_5[k][j] = %f k0\_5[k][1] =%f datatemp = %f\n",k, j, k0\_5[k][j], k0\_5[k][1]), datatemp;

if(k0\_5[k][j]> datatemp)

{

if (DE\_BUG == 1) printf("0112 XXXXXXXXXX k = %d j = %d k0\_5[k][j] = %f k0\_5[k][1] =%f\n",k, j, k0\_5[k][j], k0\_5[k][1]);

datatemp = k0\_5[k][j];

data[k][1] = data[k][j];

True\_Positives[k][1] = True\_Positives[k][j];

True\_Negatives[k][1] = True\_Negatives[k][j];

False\_Positives[k][1] = False\_Positives[k][j];

False\_Negatives[k][1] = False\_Negatives[k][j];

k0\_5[k][1] = k0\_5[k][j];

chi\_square[k][1] = chi\_square[k][j];

PVP[k][1] = PVP[k][j];

PVP\_denominator[k][1] = PVP\_denominator[k][j];

PVN[k][1] = PVN[k][j];

sensitivity[k][1] = sensitivity[k][j];

specificity[k][1] = specificity[k][j];

PVN\_denominator[k][1] = PVN\_denominator[k][j];

LT\_GE[k][1] = LT\_GE[k][j];

}

}

}

}/\*end if print long = 0\*/

if (DE\_BUG == 1) printf("0120 LOOP COMPLETED k0\_5[k][1] = %f\n",k0\_5[k][1]);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SORT 3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Print Results in ascending order for each variable value \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (DE\_BUG == 1) printf("0130 NEXT LOOP V\_Max = %d k = %d\n",variables\_max, k);

if (print\_long == 1) printf ("\*\*\*\*\*\*\*\*\*\* Largest kappa0\_5s for each variable (Mg<%d means marginal count < %d and cannot compute valid significance for Chi Square)\n", marginal\_minimum\_i, marginal\_minimum\_i);

if (print\_long == 1) printf ("\*\* Var #\tn\tCriter\tValue\tP\tQ\tsens\tspec\teffic\tk0\_0\tk0\_%2d\tk1\_0\tPVP[k][j]\tnum\tden\tPVN[k][j]\tnum\tden\tChi\_Sq\tSig\n",kappa\_level);

for (k=1;k<=variables\_max-1;k++) /\*\*\*\*\*\*for every variable \*\*\*\*\*\*\*\*/

{

if (print\_long == 1) printf("\*\* Variable = %4d ",k);

for (j=0;j<25 && var[k][j] != (int)' ';j++)

{

if (print\_long == 1) printf("%c",var[k][j]);

}

if (print\_long == 1) printf(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n",k);

for (j=1;j<2;j++) /\*\*for every value of that variable \*\*print that value generated a good selection proportion\*\*\*\*\*\*/

{

/\*\*\*Note deletion!!!if (data[k][j] != data[k][j+1] || j==cases\_max) leave out duplicate results.. save a tree\*\*\*\*\*\*\*\*\*\*\*\*/

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*Save best values for each variable separatly \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (DE\_BUG == 1) printf("0140 INSIDE LOOP j j = %d k = %d\n",j, k);

best\_value[run][k] = data[k][j];

best\_kappa[run][k] = k0\_5[k][j];

best\_chi\_square[run][k] = chi\_square[k][j];

best\_variable[run][k] = k;

best\_PVP[run][k] = PVP[k][j];

best\_PVP\_denominator[run][k] = PVP\_denominator[k][j];

best\_PVN[run][k] = PVN[k][j];

best\_sensitivity[run][k] = sensitivity[k][j];

best\_specificity[run][k] = specificity[k][j];

best\_PVN\_denominator[run][k] = PVN\_denominator[k][j];

best\_LT\_GE[run][k] = LT\_GE[k][j];

/\*printf("chi square k j = %8.3f\n",chi\_square[k][j]);\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*Need below for switch \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (DE\_BUG == 1) printf("0150 BOTTOM LOOP j j = %d k = %d\n",j, k);

/\*\* a sum of True Positives\*\*\*/

/\*\* b sum of False Positives\*\*\*/

/\*\* c sum of False Negative \*\*\*/

/\*\* d sum of True Negative \*\*\*/

aa=True\_Positives[k][j]; /\*\*predicted 1 actual 1\*\*/

bb=False\_Positives[k][j]; /\*\*predicted 1 actual 0\*\*/

cc=False\_Negatives[k][j]; /\*\*predicted 0 actual 1\*\*/

dd=True\_Negatives[k][j]; /\*\*predicted 0 actual 0\*\*/

Se = aa/(aa + bb); /\*\*\*sensitivity\*\*\*/

Sp = dd/(cc + dd); /\*\*\*specificity\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

negative\_switch = 0; /\*\*\*\*\*\*\*\*\*\*\*\* start as positive correlation \*\*\*\*\*/

if ((Se+Sp) < 1) /\*\*\*\*\*\*\*\*\*\*\*\* MAJOR SWITCH based on negative vs postive correlation of variable with gold standard \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

{

negative\_switch = 1; /\*\*\*\*\*\*\*\*\*\*\*\* change to negative correlation \*/

datatemp = aa;

aa = bb;

bb = datatemp;

datatemp = cc;

cc = dd;

dd = datatemp;

}/\*End of if Se+sp < 1\*/

aa=True\_Positives[k][j]; /\*\*predicted 1 actual 1\*\*/

bb=False\_Positives[k][j]; /\*\*predicted 1 actual 0\*\*/

cc=False\_Negatives[k][j]; /\*\*predicted 0 actual 1\*\*/

dd=True\_Negatives[k][j]; /\*\*predicted 0 actual 0\*\*/

ac=aa+cc; /\*\* actual 1 actual 0 \*\*/

ab=aa+bb; /\*\* predicted 1 aa bb ab \*\*/

bd=bb+dd; /\*\* predicted 0 cc dd cd \*\*/

cd=cc+dd; /\*\* marginal counts ac bd abcd \*\*/

abcd=aa+bb+cc+dd;

TP = aa/abcd;

FP = bb/abcd;

FN = cc/abcd;

TN = dd/abcd;

term1 = (aa-(ac\*ab/abcd)) \* (aa-(ac\*ab/abcd)) / (ac\*ab/abcd);

term2 = (bb-(bd\*ab/abcd)) \* (bb-(bd\*ab/abcd)) / (bd\*ab/abcd);

term3 = (cc-(ac\*cd/abcd)) \* (cc-(ac\*cd/abcd)) / (ac\*cd/abcd);

term4 = (dd-(bd\*cd/abcd)) \* (dd-(bd\*cd/abcd)) / (bd\*cd/abcd);

/\*\*\*termn = (cell value - expected cell value)squared / expected cell value \*\*\*/

chi\_square[k][j]= term1 + term2 + term3 + term4 ;

if (ac == 0 || ab == 0 || bd == 0 || cd == 0) chi\_square[k][j] = -9.99; /\*\* divide by zero no no \*/

P = TP+FN; /\*\*\* Prevalence - fixed \*\*\*/

P1 = 1.0-P;

Q = TP+FP; /\*\*\* Level of test- varies with test \*\*\*/

if (negative\_switch == 1) Q = (1.0-Q);

Q1 = 1.0-Q;

Se = TP/P; /\*\*\*sensitivity\*\*\*/

if (negative\_switch == 1) Se = (1.0-Se);

Sp = TN/P1; /\*\*\*specificity\*\*\*/

if (negative\_switch == 1) Sp = (1.0-Sp);

//Se, /\*\*\*sensitivity\*\*\*/

//Sp, /\*\*\*specificity\*\*\*/

sensitivity[k][j] = Se,

specificity[k][j] = Sp,

PVP[k][j] = aa/ab;

if (ab == 0.0) PVP[k][j]= -9.99;

/\*\*\*\*\*PVP = TP/Q; cannot get to work \*\*\*\*\*/

PVN[k][j] = dd/cd;

if (cd == 0.0) PVN[k][j]= -9.99;

/\*PVN = TN/Q1; cannot get to work \*\*\*\*\*/

if (Q == 0) PVP[k][j] = -9.99; /\*\* identify division by zero cases \*\*/

if (Q1 == 0) PVN[k][j] = -9.99; /\*\* identify division by zero cases \*\*/

Eff = TP+TN; /\*\*\*efficiency\*\*\*/

if (negative\_switch == 1) Eff = (1.0-Eff);

k1\_0 = (Se-Q)/Q1;

if (Q1 == 0) k1\_0 = -9.99;

k0\_0 = (Sp-Q1)/Q;

if (Q == 0) k0\_0 = -9.99;

/\*k0\_5[k][j] = (Eff - P\*Q - P1\*Q1)/(1.0 - P\*Q - P1\*Q1); \*/

/\*error of 4.15 take out this line to carry forward k0\_5 calculation done above \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\* printing only below here \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (DE\_BUG == 1) printf("0160 PRINTING ONLY BELOW j = %d k = %d\n",j, k);

if (negative\_switch == 0)

{

if (DE\_BUG == 1) printf("0170 negative\_switch == 0 j = %d k = %d\n",j, k);

if (ab >= marginal\_minimum && cd >= marginal\_minimum && ac >= marginal\_minimum && bd >= marginal\_minimum)

{

if (DE\_BUG == 1) printf ("Print Number 3 Neg switch ==0\n"); /\*\*/

if (print\_long == 1) printf ("%d\t%5.0f\tGE\t%8.3f\t%5.3f\t%5.3f\t%5.3f\t%5.3f\t%5.3f\t%5.3f\t%5.3f\t%5.3f\t%4.2f\t%4.0f\t%4.0f\t%4.2f\t%5.0f\t%5.0f\t%6.2f\t"

,k,variables\_n[k],data[k][j],

P,

Q,

//Se, /\*\*\*sensitivity\*\*\*/

//Sp, /\*\*\*specificity\*\*\*/

sensitivity[k][j],

specificity[k][j],

Eff, /\*\*\*efficiency\*\*\*/

k0\_0,

k0\_5[k][j] ,

k1\_0,

PVP[k][j],

aa,ab,

PVN[k][j],

dd,cd,

chi\_square[k][j]);

if (ab >= marginal\_minimum && cd >= marginal\_minimum && ac >= marginal\_minimum && bd >= marginal\_minimum)

{

if (chi\_square[k][j] >= 3.84 && chi\_square[k][j] < 6.64) if (print\_long == 1) printf("\*"); /\*\*p <.05 etc.\*\*/

if (chi\_square[k][j] >= 6.64 && chi\_square[k][j] < 10.83) if (print\_long == 1) printf("\*\*"); /\*\*p <.01 etc.\*\*/

if (chi\_square[k][j] >= 10.83) if (print\_long == 1) printf("\*\*\*"); /\*\*p <.001 etc.\*\*/

}

else

{

if (chi\_square[k][j] == -9.99)

{

if (print\_long == 1) printf("Cell=0"); /\*\*division by zero\*\*/

}

else

{

if (print\_long == 1) printf("Cell<10"); /\*\*too few cells for accurate Chi Square\*\*/

}

}

if (print\_long == 1) printf("\n");

}/\*\*\* if (ab > 9 && cd > 9 && ac > 9 && bd > 9) \*/

}/\*\*\*\*\*end if negative\_switch == 0 \*/

if (negative\_switch == 1)

{

if (DE\_BUG == 1) printf("0180 negative\_switch == 1 j = %d k = %d\n",j, k);

PVP[k][j] = cc/cd;

if (cd == 0) PVP[k][j] = -9.99;

PVN[k][j] = bb/ab;

if (ab == 0) PVN[k][j] = -9.99;

if (ab >= marginal\_minimum && cd >= marginal\_minimum && ac >= marginal\_minimum && bd >= marginal\_minimum)

{

if (DE\_BUG == 1) printf ("Print Number 4 Neg switch ==1\n"); /\*\*/

if (print\_long == 1) printf ("%d\t%5.0f\tLT\t%8.2f\t%5.3f\t%5.3f\t%5.3f\t%5.3f\t%5.3f\t%5.3f\t%5.3f\t%5.3f\t%4.2f\t%4.0f\t%4.0f\t%4.2f\t%5.0f\t%5.0f\t%6.2f\t"

,k,variables\_n[k],data[k][j],

P,

Q,

//Se, /\*\*\*sensitivity\*\*\*/

//Sp, /\*\*\*specificity\*\*\*/

sensitivity[k][j],

specificity[k][j],

Eff, /\*\*\*efficiency\*\*\*/

k0\_0,

k0\_5[k][j] ,

k1\_0,

PVP[k][j],

dd,cd,

PVN[k][j],

bb,ab,

chi\_square[k][j]);

if (ab >= marginal\_minimum && cd >= marginal\_minimum && ac >= marginal\_minimum && bd >= marginal\_minimum)

{

if (chi\_square[k][j] >= 3.84 && chi\_square[k][j] < 6.64) if (print\_long == 1) printf("\*"); /\*\*p <.05 etc.\*\*/

if (chi\_square[k][j] >= 6.64 && chi\_square[k][j] < 10.83) if (print\_long == 1) printf("\*\*"); /\*\*p <.01 etc.\*\*/

if (chi\_square[k][j] >= 10.83) if (print\_long == 1) printf("\*\*\*"); /\*\*p <.001 etc.\*\*/

}

else

{

if (chi\_square[k][j] == -9.99)

{

if (print\_long == 1) printf("Cell=0"); /\*\*division by zero\*\*/

}

else

{

if (print\_long == 1) printf("Cell<5"); /\*\*too few cells for accurate Chi Square\*\*/

}

}

if (print\_long == 1) printf("\n");

}/\*\*\*if (ab > 9 && cd > 9 && ac > 9 && bd > 9)\*\*\*/

}/\*\*\*\*\*end if negative\_switch == 0 \*/

}/\*End of if (data[k][j] != data[k][j+1] || j==cases\_max) /\*\*\*leave out duplicate results.. save a tree\*\*\*/

}/\*End of for (j=1;j<=cases\_max;j++) /\*\*for every value of that variable \*\*print that value generated a good selection proportion\*\*\*\*\*\*/

}/\*End of for (k=1;k<=variables\_max-1;k++) /\*\*\*\*\*\*for every variable \*\*\*\*\*\*\*\*/

}/\*End of if (plot == 0) Do this only if not a plot data run\*/

for (j=1;j<=variables\_max-1;j++)

{

if(best\_kappa[run][j]>best\_kappa[run][1])

{

if (DE\_BUG == 1) printf("0300 PRINT BEST VALUES j = %d\n",j);

best\_kappa[run][1] = best\_kappa[run][j];

best\_variable[run][1] = best\_variable[run][j];

best\_value[run][1] = best\_value[run][j];

best\_chi\_square[run][1] = best\_chi\_square[run][j];

best\_PVP[run][1] = best\_PVP[run][j];

best\_PVP\_denominator[run][1] = best\_PVP\_denominator[run][j];

best\_PVN[run][1] = best\_PVN[run][j];

best\_sensitivity[run][1] = best\_sensitivity[run][j];

best\_specificity[run][1] = best\_specificity[run][j];

best\_PVN\_denominator[run][1] = best\_PVN\_denominator[run][j];

best\_LT\_GE[run][1] = best\_LT\_GE[run][j];

}

}

separators();

if (DE\_BUG == 1) printf("0210 j = %d k = %d\n",j, k);

for (j=1;j<2;j++)

{

if (best\_LT\_GE[run][j] == 0.0) printf("\*\*Best Variable = %5.3f at cut\_value LT = %5.3f kappa = %5.3f chi square = %8.3f\n",

best\_variable[run][j],best\_value[run][j],best\_kappa[run][j],best\_chi\_square[run][j]);

if (best\_LT\_GE[run][j] == 1.0) printf("\*\*Best Variable = %5.3f at cut\_value GE = %5.3f kappa = %5.3f chi square = %8.3f\n",

best\_variable[run][j],best\_value[run][j],best\_kappa[run][j],best\_chi\_square[run][j]);

}

separators();

} /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*this was/is last parentheses \*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double sdndev(double sumsq,double sumrt,double n)

{

double z, sdev;

z = 0;

sdev = -99999;

if (sumsq == 0) z = 1;

if (sumrt == 0) z = 2;

if (n == 0) z = 3;

if (z == 0) sdev = (sumsq - ((sumrt\*sumrt)/n))/(n-1);

if (sdev < 0) sdev = sdev \* -1;

sdev = sqrt(sdev);

if (z == 1) sdev = -99;

if (z == 2) sdev = -999;

if (z == 3) sdev = -9999;

return sdev;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void zero\_variables()

{

for (k=0;k<(variables\_max+1);k++)

{

variables\_sum[k]=variables\_mean[k]=variables\_n[k]=variables\_sumsq[k]=variables\_sumrt[k]=0.0;

datatemp = 0.0;

variables\_maximum[k] = 0.0;

variables\_minimum[k]=1000000.0;

for (j=0;j<cases\_max+1;j=j+1)

{

k0\_5[k][j] = 0.0;

chi\_square[k][j] = 0.0;

False\_Positives[k][j] = 0.0;

False\_Negatives[k][j] = 0.0;

True\_Positives[k][j] = 0.0;

True\_Negatives[k][j] = 0.0;

PVP[k][j] = 0.0;

PVN[k][j] = 0.0;

sensitivity[k][j]=0.0;

specificity[k][j]=0.0;

PVP\_denominator[k][j] = 0.0;

PVN\_denominator[k][j] = 0.0;

LT\_GE[k][j] = 0;

}

}

i=j=k=l=m=n=variables=cases=letters=0;

f = oldf = 0.0;

aa=bb=cc=dd=ac=ab=bd=cd=abcd=term1=term2=term3=term4=0.0;

P = P1 = Eff = Sp = Se = Q = Q1 = k0\_0 = k1\_0 = 0.0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void separators()

{

printf("\*\*\n"); /\*like to put separators here \*/

printf("\*\*\n"); /\*like to put separators here \*/

printf("\*\*\n"); /\*like to put separators here \*/

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void recover\_data()

{

for (k=0;k<(variables\_max+1);k++)

{

for (j=0;j<cases\_max+1;j=j+1)

{

data[k][j] = data2[k][j];

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void print\_kappa()

{

for (k=1;k<=variables\_max-1;k++)

{

for (j=1;j<=cases\_max-1;j++) printf("k = %d j = %d k0\_5[k][j] = %5.3f chi\_square[k][j] = %5.3f\n",

k,j,k0\_5[k][j],chi\_square[k][j]);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void print\_data()

{

for (k=1;k<=variables\_max-1;k++)

{

for (j=1;j<=cases\_max-1;j++) printf("k = %d j = %d data[k][j] = %5.3f\n",

k,j,data[k][j]);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void remove\_GE\_Data(int run2)

{

for (j=0;j<cases\_max+1;j=j+1) /\*eliminate data for values GE cut point \*/

{

//if (data[(int)best\_variable[run2][1]][j] >= best\_value[run2][1] || (data[(int)best\_variable[run2][1]][j] == MISSING\_INT) || (data[(int)best\_variable[run2][1]][j] == MISSING\_FLOAT))

if (data[(int)best\_variable[run2][1]][j] >= best\_value[run2][1] || data[(int)best\_variable[run2][1]][j] == MISSING\_INT)

{

for (k=0;k<(variables\_max+1);k++)

{

data[k][j] = MISSING\_INT;

}

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void remove\_LT\_Data(int run3)

{

for (j=0;j<cases\_max+1;j=j+1) /\*eliminate data for values LT cut point \*/

{

if (data[(int)best\_variable[run3][1]][j] < best\_value[run3][1] || data[(int)best\_variable[run3][1]][j] == MISSING\_INT)

{

for (k=0;k<(variables\_max+1);k++)

{

data[k][j] = MISSING\_INT;

}

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void dimension\_array\_size()

{

data = ((float \*\*)malloc((variables\_max+2)\*sizeof(float \*))); /\* add 1 as variables start as #1 not #0\*/

if( data == NULL )fprintf(stdout,"malloc error data1\n" );

for (n=0; n < variables\_max+2; n++)

{

data[n] = (float \*)malloc((cases\_max+2)\*sizeof(float ));

if( data[n] == NULL )fprintf(stdout,"malloc error data2\n" );

}

//char \*\*var[NCOL][25];

var = ((char \*\*)malloc((variables\_max+3)\*sizeof(char \*)));

if( var == NULL )

{

fprintf(stdout,"malloc error var1\n" );

}

for (n=0; n < (variables\_max+3); n++)

{

var[n] = (char \*)malloc((25)\*sizeof(char ));

if( var[n] == NULL )

{

fprintf(stdout,"malloc error var2\n" );

}

}

//float best\_value[NRUN][NCOL], best\_kappa[NRUN][NCOL], best\_variable[NRUN][NCOL], best\_chi\_square[NRUN][NCOL];

best\_value = ((float \*\*)malloc((NRUN+1)\*sizeof(float \*)));

if( best\_value == NULL )fprintf(stdout,"malloc error best\_value1\n" );

for (n=0; n < (NRUN+1); n++)

{

best\_value[n] = (float \*)malloc((variables\_max+3)\*sizeof(float ));

if( best\_value[n] == NULL )fprintf(stdout,"malloc error best\_value2\n" );

}

best\_kappa = ((float \*\*)malloc((NRUN+1)\*sizeof(float \*)));

if( best\_kappa == NULL )

{

fprintf(stdout,"malloc error best\_kappa1\n" );

}

for (n=0; n < (NRUN+1); n++)

{

best\_kappa[n] = (float \*)malloc((variables\_max+2)\*sizeof(float ));

if( best\_kappa[n] == NULL )

{

fprintf(stdout,"malloc error best\_kappa2\n" );

}

}

best\_variable = ((float \*\*)malloc((NRUN+1)\*sizeof(float \*)));

if( best\_variable == NULL )

{

fprintf(stdout,"malloc error best\_variable1\n" );

}

for (n=0; n < (NRUN+1); n++)

{

best\_variable[n] = (float \*)malloc((variables\_max+2)\*sizeof(float ));

if( best\_variable[n] == NULL )

{

fprintf(stdout,"malloc error best\_variable2\n" );

}

}

best\_chi\_square = ((float \*\*)malloc((NRUN+1)\*sizeof(float \*)));

if( best\_chi\_square == NULL )

{

fprintf(stdout,"malloc error best\_chi\_square1\n" );

}

for (n=0; n < (NRUN+1); n++)

{

best\_chi\_square[n] = (float \*)malloc((variables\_max+2)\*sizeof(float ));

if( best\_chi\_square[n] == NULL )

{

fprintf(stdout,"malloc error best\_chi\_square2\n" );

}

}

//float best\_PVP[NRUN][NCOL], best\_PVP\_denominator[NRUN][NCOL], best\_PVN[NRUN][NCOL], best\_PVN\_denominator[NRUN][NCOL], best\_LT\_GE[NRUN][NCOL],\*\*best\_sensitivity, \*\*best\_specificity;

best\_PVP = ((float \*\*)malloc((NRUN+1)\*sizeof(float \*)));

if( best\_PVP == NULL )

{

fprintf(stdout,"malloc error best\_PVP1\n" );

}

for (n=0; n < (NRUN+1); n++)

{

best\_PVP[n] = (float \*)malloc((variables\_max+2)\*sizeof(float ));

if( best\_PVP[n] == NULL )

{

fprintf(stdout,"malloc error best\_PVP2\n" );

}

}

//float best\_PVP[NRUN][NCOL], best\_PVP\_denominator[NRUN][NCOL], best\_PVN[NRUN][NCOL], best\_PVN\_denominator[NRUN][NCOL], best\_LT\_GE[NRUN][NCOL],\*\*best\_sensitivity, \*\*best\_specificity;

best\_sensitivity = ((float \*\*)malloc((NRUN+1)\*sizeof(float \*)));

if( best\_sensitivity == NULL )

{

fprintf(stdout,"malloc error best\_sensitivity\n" );

}

for (n=0; n < (NRUN+1); n++)

{

best\_sensitivity[n] = (float \*)malloc((variables\_max+2)\*sizeof(float ));

if( best\_sensitivity[n] == NULL )

{

fprintf(stdout,"malloc error best\_sensitivity\n" );

}

}

//float best\_PVP[NRUN][NCOL], best\_PVP\_denominator[NRUN][NCOL], best\_PVN[NRUN][NCOL], best\_PVN\_denominator[NRUN][NCOL], best\_LT\_GE[NRUN][NCOL],\*\*best\_sensitivity, \*\*best\_specificity;

best\_specificity = ((float \*\*)malloc((NRUN+1)\*sizeof(float \*)));

if( best\_specificity == NULL )

{

fprintf(stdout,"malloc error best\_specificity\n" );

}

for (n=0; n < (NRUN+1); n++)

{

best\_specificity[n] = (float \*)malloc((variables\_max+2)\*sizeof(float ));

if( best\_specificity[n] == NULL )

{

fprintf(stdout,"malloc error best\_specificity\n" );

}

}

best\_PVP\_denominator = ((float \*\*)malloc((NRUN+1)\*sizeof(float \*)));

if( best\_PVP\_denominator == NULL )

{

fprintf(stdout,"malloc error best\_PVP\_denominator1\n" );

}

for (n=0; n < (NRUN+1); n++)

{

best\_PVP\_denominator[n] = (float \*)malloc((variables\_max+2)\*sizeof(float ));

if( best\_PVP\_denominator[n] == NULL )

{

fprintf(stdout,"malloc error best\_PVP\_denominator2\n" );

}

}

best\_PVN = ((float \*\*)malloc((NRUN+1)\*sizeof(float \*)));

if( best\_PVN == NULL )

{

fprintf(stdout,"malloc error best\_PVN1\n" );

}

for (n=0; n < (NRUN+1); n++)

{

best\_PVN[n] = (float \*)malloc((variables\_max+2)\*sizeof(float ));

if( best\_PVN[n] == NULL )

{

fprintf(stdout,"malloc error best\_PVN2\n" );

}

}

best\_PVN\_denominator = ((float \*\*)malloc((NRUN+1)\*sizeof(float \*)));

if( best\_PVN\_denominator == NULL )

{

fprintf(stdout,"malloc error best\_PVN\_denominator1\n" );

}

for (n=0; n < (NRUN+1); n++)

{

best\_PVN\_denominator[n] = (float \*)malloc((variables\_max+2)\*sizeof(float ));

if( best\_PVN\_denominator[n] == NULL )

{

fprintf(stdout,"malloc error best\_PVN\_denominator2\n" );

}

}

best\_LT\_GE = ((float \*\*)malloc((NRUN+1)\*sizeof(float \*)));

if( best\_LT\_GE == NULL )

{

fprintf(stdout,"malloc error best\_LT\_GE1\n" );

}

for (n=0; n < (NRUN+1); n++)

{

best\_LT\_GE[n] = (float \*)malloc((variables\_max+2)\*sizeof(float ));

if( best\_LT\_GE[n] == NULL )

{

fprintf(stdout,"malloc error best\_LT\_GE2\n" );

}

}

//float variables\_maximum[NCOL],variables\_minimum[NCOL];

variables\_maximum = ((float \*)malloc((variables\_max+2)\*sizeof(float \*)));

if(variables\_maximum == NULL )

{

fprintf(stdout,"malloc error variables\_maximum\n" );

}

variables\_minimum = ((float \*)malloc((variables\_max+2)\*sizeof(float \*)));

if(variables\_minimum == NULL )

{

fprintf(stdout,"malloc error variables\_minimum\n" );

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* New Variables \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*float variables\_sum[NCOL],variables\_mean[NCOL],variables\_n[NCOL],variables\_sumsq[NCOL],variables\_sumrt[NCOL];

float variables\_maximum[NCOL],variables\_minimum[NCOL];\*/

variables\_sum = ((float \*)malloc((variables\_max+2)\*sizeof(float \*)));

if(variables\_sum == NULL )

{

fprintf(stdout,"malloc error variables\_sum\n" );

}

variables\_mean = ((float \*)malloc((variables\_max+2)\*sizeof(float \*)));

if(variables\_mean == NULL )

{

fprintf(stdout,"malloc error variables\_mean\n" );

}

variables\_n = ((float \*)malloc((variables\_max+2)\*sizeof(float \*)));

if(variables\_n == NULL )

{

fprintf(stdout,"malloc error variables\_n\n" );

}

variables\_sumsq = ((float \*)malloc((variables\_max+2)\*sizeof(float \*)));

if(variables\_sumsq == NULL )

{

fprintf(stdout,"malloc error variables\_sumsq\n" );

}

variables\_sumrt = ((float \*)malloc((variables\_max+2)\*sizeof(float \*)));

if(variables\_sumrt == NULL )

{

fprintf(stdout,"malloc error variables\_sumrt\n" );

}

False\_Positives = ((short \*\*)malloc((variables\_max+2)\*sizeof(short \*)));

if( False\_Positives == NULL )

{

fprintf(stdout,"malloc error False\_Positives1\n" );

}

for (n=0; n < (variables\_max+2); n++)

{

False\_Positives[n] = (short \*)malloc((cases\_max+2)\*sizeof(short ));

if( False\_Positives[n] == NULL )

{

fprintf(stdout,"malloc error False\_Positives2\n" );

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

False\_Negatives = ((short \*\*)malloc((variables\_max+2)\*sizeof(short \*)));

if( False\_Negatives == NULL )

{

fprintf(stdout,"malloc error False\_Negatives1\n" );

}

for (n=0; n < (variables\_max+2); n++)

{

False\_Negatives[n] = (short \*)malloc((cases\_max+2)\*sizeof(short ));

if( False\_Negatives[n] == NULL )

{

fprintf(stdout,"malloc error False\_Negatives2\n" );

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

True\_Positives = ((short \*\*)malloc((variables\_max+2)\*sizeof(short \*)));

if(True\_Positives == NULL )

{

fprintf(stdout,"malloc error True\_Positives1\n" );

}

for (n=0; n < (variables\_max+2); n++)

{

True\_Positives[n] = (short \*)malloc((cases\_max+2)\*sizeof(short ));

if( True\_Positives[n] == NULL )

{

fprintf(stdout,"malloc error True\_Positives2\n" );

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

True\_Negatives = ((short \*\*)malloc((variables\_max+2)\*sizeof(short \*)));

if( True\_Negatives == NULL )

{

fprintf(stdout,"malloc error True\_Negatives1\n" );

}

for (n=0; n < (variables\_max+2); n++)

{

True\_Negatives[n] = (short \*)malloc((cases\_max+2)\*sizeof(short ));

if( True\_Negatives[n] == NULL )

{

fprintf(stdout,"malloc error True\_Negatives2\n" );

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

chi\_square = ((float \*\*)malloc((variables\_max+2)\*sizeof(float \*)));

if( chi\_square == NULL )

{

fprintf(stdout,"malloc error chi\_square1\n" );

}

for (n=0; n < (variables\_max+2); n++)

{

chi\_square[n] = (float \*)malloc((cases\_max+2)\*sizeof(float ));

if( chi\_square[n] == NULL )

{

fprintf(stdout,"malloc error chi\_square2\n" );

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

k0\_5 = ((float \*\*)malloc((variables\_max+2)\*sizeof(float \*)));

if( k0\_5 == NULL )

{

fprintf(stdout,"malloc error k0\_5 1\n" );

}

for (n=0; n < (variables\_max+2); n++)

{

k0\_5[n] = (float \*)malloc((cases\_max+2)\*sizeof(float ));

if( k0\_5[n] == NULL )

{

fprintf(stdout,"malloc error k0\_5 2\n" );

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

PVP = ((float \*\*)malloc((variables\_max+2)\*sizeof(float \*)));

if( PVP == NULL )

{

fprintf(stdout,"malloc error PVP 1\n" );

}

for (n=0; n < (variables\_max+2); n++)

{

PVP[n] = (float \*)malloc((cases\_max+2)\*sizeof(float ));

if( PVP[n] == NULL )

{

fprintf(stdout,"malloc error PVP 2\n" );

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

PVN = ((float \*\*)malloc((variables\_max+2)\*sizeof(float \*)));

if( PVN == NULL )

{

fprintf(stdout,"malloc error PVN 1\n" );

}

for (n=0; n < (variables\_max+2); n++)

{

PVN[n] = (float \*)malloc((cases\_max+2)\*sizeof(float ));

if( PVN[n] == NULL )

{

fprintf(stdout,"malloc error PVN 2\n" );

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

PVP\_denominator = ((short \*\*)malloc((variables\_max+2)\*sizeof(short \*)));

if( PVP\_denominator == NULL )

{

fprintf(stdout,"malloc error PVP\_denominator 1\n" );

}

for (n=0; n < (variables\_max+2); n++)

{

PVP\_denominator[n] = (short \*)malloc((cases\_max+2)\*sizeof(short ));

if( PVP\_denominator[n] == NULL )

{

fprintf(stdout,"malloc error PVP\_denominator 2\n" );

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

PVN\_denominator = ((short \*\*)malloc((variables\_max+2)\*sizeof(short \*)));

if( PVN\_denominator == NULL )

{

fprintf(stdout,"malloc error PVN\_denominator 1\n" );

}

for (n=0; n < (variables\_max+2); n++)

{

PVN\_denominator[n] = (short \*)malloc((cases\_max+2)\*sizeof(short ));

if( PVN\_denominator[n] == NULL )

{

fprintf(stdout,"malloc error PVN\_denominator 2\n" );

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

LT\_GE = ((\_\_int8 \*\*)malloc((variables\_max+2)\*sizeof(\_\_int8 \*)));

if( LT\_GE == NULL )

{

fprintf(stdout,"malloc error LT\_GE 1\n" );

}

for (n=0; n < (variables\_max+2); n++)

{

LT\_GE[n] = (\_\_int8 \*)malloc((cases\_max+2)\*sizeof(\_\_int8 ));

if( LT\_GE[n] == NULL )

{

fprintf(stdout,"malloc error LT\_GE 2\n" );

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

data2 = ((float \*\*)malloc((variables\_max+2)\*sizeof(float \*)));

if( data2 == NULL )

{

fprintf(stdout,"malloc error data2 1\n" );

}

for (n=0; n < (variables\_max+2); n++)

{

data2[n] = (float \*)malloc((cases\_max+2)\*sizeof(float ));

if( data2[n] == NULL )

{

fprintf(stdout,"malloc error data2 2\n" );

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

sensitivity = ((float \*\*)malloc((variables\_max+2)\*sizeof(float \*)));

if( sensitivity == NULL )

{

fprintf(stdout,"malloc error sensitivity 1\n" );

}

for (n=0; n < (variables\_max+2); n++)

{

sensitivity[n] = (float \*)malloc((cases\_max+2)\*sizeof(float ));

if( sensitivity[n] == NULL )

{

fprintf(stdout,"malloc error sensitivity 2\n" );

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

specificity = ((float \*\*)malloc((variables\_max+2)\*sizeof(float \*)));

if( specificity == NULL )

{

fprintf(stdout,"malloc error specificity 1\n" );

}

for (n=0; n < (variables\_max+2); n++)

{

specificity[n] = (float \*)malloc((cases\_max+2)\*sizeof(float ));

if( specificity[n] == NULL )

{

fprintf(stdout,"malloc error specificity 2\n" );

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void print\_variable\_names(int var\_number)

{

for (m=1;m<25;m++) printf("%c",var[var\_number][m]);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void print\_variable\_names\_short(int var\_number)

{

for (m=1;m<25;m++) printf("%c",var[var\_number][m]);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void print\_variable\_names\_eight(int var\_number)

{

for (m=1;m<9;m++) printf("%c",var[var\_number][m]);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void print\_variable\_names\_eight\_blank(int var\_number)

{

for (m=1;m<9;m++) printf(" ");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void print\_long\_line()

{

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n",k);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void print\_summary()

{

if (DE\_BUG == 1) printf("1000 COMMAND LINE kappa\_level2 = %d plot3 = %d print\_long4 = %d DE\_BUG5 = %d p\_value\_to\_progress6 = %f\n",kappa\_level, plot, print\_long, DE\_BUG, p\_value\_to\_progress);

printf("\*\*\*Summary:\n");

printf("\*\*\*Note on Significance: \*\*\* is p <.001 \*\* is p <.01 for intermediate results a \* is p <.05\n");

printf("\*\*\*Note on Significance: \*\*\* summary runs only performed if p < %5.3f\n", p\_value\_to\_progress);

printf("\*\*\*Run Number\t\tBest Variable\t\t\tCut Point\tPVP\tPVP n\tPVN\tPVN n\tSens\tSpec\tkappa\tchi square\n");

for (n = 1; n <= NRUN; n++)

{

if (DE\_BUG == 1) printf ("NRUN = %5d \tLT\t%8.3f\t%5.3f\t%5d\t%5.3f\t%5d\t%5.3f\t%8.3f\n",

n, best\_value[n][1], best\_PVP[n][1], (int)best\_PVP\_denominator[n][1], best\_PVN[n][1], (int)best\_PVN\_denominator[n][1], best\_kappa[n][1], best\_chi\_square[n][1]);

}

for (n = 1; n <= NRUN; n++)

{

if ((int)best\_variable[n][1] != 0)

{

//if (best\_value[n][1] == -9999.99) best\_value[n][1] = 0.0;

//if (best\_value[n][1] == -9999.00) best\_value[n][1] = 0.0;

printf ("\*\*\*\t%2d\t%2d\t",n,(int)best\_variable[n][1]);

print\_variable\_names\_short((int)best\_variable[n][1]);

if (best\_LT\_GE[n][1] == 0.0) printf ("\tLT\t%8.3f\t%5.3f\t%5d\t%5.3f\t%5d\t%5.3f\t%5.3f\t%5.3f\t%8.3f",

best\_value[n][1], best\_PVP[n][1], (int)best\_PVP\_denominator[n][1], best\_PVN[n][1], (int)best\_PVN\_denominator[n][1], best\_sensitivity[n][1], best\_specificity[n][1], best\_kappa[n][1], best\_chi\_square[n][1]);

if (best\_LT\_GE[n][1] == 1.0) printf ("\tGE\t%8.3f\t%5.3f\t%5d\t%5.3f\t%5d\t%5.3f\t%5.3f\t%5.3f\t%8.3f",

best\_value[n][1], best\_PVP[n][1], (int)best\_PVP\_denominator[n][1], best\_PVN[n][1], (int)best\_PVN\_denominator[n][1], best\_sensitivity[n][1], best\_specificity[n][1], best\_kappa[n][1], best\_chi\_square[n][1]);

if (best\_chi\_square[n][1] >= 10.83) printf("\t\*\*\* \t");

if (best\_chi\_square[n][1] >= 6.64 && best\_chi\_square[n][1] < 10.83) printf("\t\*\* \t");

if (best\_chi\_square[n][1] >= 3.84 && best\_chi\_square[n][1] < 6.64) printf("\t\* \t");

if (n == 1 && best\_chi\_square[1][1] >= 10.827) printf("Run 1 Merits Runs 2&3");

if (n == 1 && best\_chi\_square[1][1] >= 6.64 && best\_chi\_square[n][1] < 10.827) printf("Run 1 Merits Runs 2&3");

if (n == 1 && best\_chi\_square[1][1] >= 3.84 && best\_chi\_square[n][1] < 6.64 && p\_value\_to\_progress > .01) printf("Run 1 Merits Runs 2&3");

if (n == 2 && best\_chi\_square[2][1] >= 10.827) printf("Run 2 Merits Runs 4&5");

if (n == 2 && best\_chi\_square[2][1] >= 6.64 && best\_chi\_square[n][1] < 10.827) printf("Run 2 Merits Runs 4&5");

if (n == 2 && best\_chi\_square[2][1] >= 3.84 && best\_chi\_square[n][1] < 6.64 && p\_value\_to\_progress > .01) printf("Run 2 Merits Runs 4&5");

if (n == 3 && best\_chi\_square[3][1] >= 10.827) printf("Run 3 Merits Runs 6&7");

if (n == 3 && best\_chi\_square[3][1] >= 6.64 && best\_chi\_square[n][1] < 10.83) printf("Run 3 Merits Runs 6&7");

if (n == 3 && best\_chi\_square[3][1] >= 3.84 && best\_chi\_square[n][1] < 6.64 && p\_value\_to\_progress > .01) printf("Run 3 Merits Runs 6&7");

if (n == 4 && best\_chi\_square[4][1] >= 10.827) printf("Run 4 Merits Runs 8&9");

if (n == 4 && best\_chi\_square[4][1] >= 6.64 && best\_chi\_square[n][1] < 10.827) printf("Run 4 Merits Runs 8&9");

if (n == 4 && best\_chi\_square[4][1] >= 3.84 && best\_chi\_square[n][1] < 6.64 && p\_value\_to\_progress > .01) printf("Run 4 Merits Runs 8&9");

if (n == 5 && best\_chi\_square[5][1] >= 10.827) printf("Run 5 Merits Runs 10&11");

if (n == 5 && best\_chi\_square[5][1] >= 6.64 && best\_chi\_square[n][1] < 10.827) printf("Run 5 Merits Runs 10&11");

if (n == 5 && best\_chi\_square[5][1] >= 3.84 && best\_chi\_square[n][1] < 6.64 && p\_value\_to\_progress > .01) printf("Run 5 Merits Runs 10&11");

if (n == 6 && best\_chi\_square[6][1] >= 10.827) printf("Run 6 Merits Runs 12&13");

if (n == 6 && best\_chi\_square[6][1] >= 6.64 && best\_chi\_square[n][1] < 10.827) printf("Run 6 Merits Runs 12&13");

if (n == 6 && best\_chi\_square[6][1] >= 3.84 && best\_chi\_square[n][1] < 6.64 && p\_value\_to\_progress > .01) printf("Run 6 Merits Runs 12&13");

if (n == 7 && best\_chi\_square[7][1] >= 10.827) printf("Run 7 Merits Runs 14&15");

if (n == 7 && best\_chi\_square[7][1] >= 6.64 && best\_chi\_square[n][1] < 10.827) printf("Run 7 Merits Runs 14&15");

if (n == 7 && best\_chi\_square[7][1] >= 3.84 && best\_chi\_square[n][1] < 6.64 && p\_value\_to\_progress > .01) printf("Run 7 Merits Runs 14&15");

if (best\_value[n][1] == -9999.00) printf("\tOne marginal n < %d cannot compute", marginal\_minimum\_i);

printf ("\n");

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void print\_diagram()

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Set variable to zero as marker of small chi squares\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (DE\_BUG == 1) printf("2000 COMMAND LINE kappa\_level2 = %d plot3 = %d print\_long4 = %d DE\_BUG5 = %d p\_value\_to\_progress6 = %f\n",kappa\_level, plot, print\_long, DE\_BUG, p\_value\_to\_progress);

if (p\_value\_to\_progress == .05)

{

if (best\_chi\_square[1][1] < 3.84) best\_PVP[1][1] = best\_PVN[1][1] = 0.0;

if (best\_chi\_square[2][1] < 3.84) best\_PVP[2][1] = best\_PVN[2][1] = 0.0;

if (best\_chi\_square[3][1] < 3.84) best\_PVP[3][1] = best\_PVN[3][1] = 0.0;

if (best\_chi\_square[4][1] < 3.84) best\_PVP[4][1] = best\_PVN[4][1] = 0.0;

if (best\_chi\_square[5][1] < 3.84) best\_PVP[5][1] = best\_PVN[5][1] = 0.0;

if (best\_chi\_square[6][1] < 3.84) best\_PVP[6][1] = best\_PVN[6][1] = 0.0;

if (best\_chi\_square[7][1] < 3.84) best\_PVP[7][1] = best\_PVN[7][1] = 0.0;

if (best\_chi\_square[8][1] < 3.84) best\_PVP[8][1] = best\_PVN[8][1] = 0.0;

if (best\_chi\_square[9][1] < 3.84) best\_PVP[9][1] = best\_PVN[9][1] = 0.0;

if (best\_chi\_square[10][1] < 3.84) best\_PVP[10][1] = best\_PVN[10][1] = 0.0;

if (best\_chi\_square[11][1] < 3.84) best\_PVP[11][1] = best\_PVN[11][1] = 0.0;

if (best\_chi\_square[12][1] < 3.84) best\_PVP[12][1] = best\_PVN[12][1] = 0.0;

if (best\_chi\_square[13][1] < 3.84) best\_PVP[13][1] = best\_PVN[13][1] = 0.0;

if (best\_chi\_square[14][1] < 3.84) best\_PVP[14][1] = best\_PVN[14][1] = 0.0;

if (best\_chi\_square[15][1] < 3.84) best\_PVP[15][1] = best\_PVN[15][1] = 0.0;

}

if (p\_value\_to\_progress == .01)

{

if (best\_chi\_square[1][1] < 6.636) best\_PVP[1][1] = best\_PVN[1][1] = 0.0;

if (best\_chi\_square[2][1] < 6.636) best\_PVP[2][1] = best\_PVN[2][1] = 0.0;

if (best\_chi\_square[3][1] < 6.636) best\_PVP[3][1] = best\_PVN[3][1] = 0.0;

if (best\_chi\_square[4][1] < 6.636) best\_PVP[4][1] = best\_PVN[4][1] = 0.0;

if (best\_chi\_square[5][1] < 6.636) best\_PVP[5][1] = best\_PVN[5][1] = 0.0;

if (best\_chi\_square[6][1] < 6.636) best\_PVP[6][1] = best\_PVN[6][1] = 0.0;

if (best\_chi\_square[7][1] < 6.636) best\_PVP[7][1] = best\_PVN[7][1] = 0.0;

if (best\_chi\_square[8][1] < 6.636) best\_PVP[8][1] = best\_PVN[8][1] = 0.0;

if (best\_chi\_square[9][1] < 6.636) best\_PVP[9][1] = best\_PVN[9][1] = 0.0;

if (best\_chi\_square[10][1] < 6.636) best\_PVP[10][1] = best\_PVN[10][1] = 0.0;

if (best\_chi\_square[11][1] < 6.636) best\_PVP[11][1] = best\_PVN[11][1] = 0.0;

if (best\_chi\_square[12][1] < 6.636) best\_PVP[12][1] = best\_PVN[12][1] = 0.0;

if (best\_chi\_square[13][1] < 6.636) best\_PVP[13][1] = best\_PVN[13][1] = 0.0;

if (best\_chi\_square[14][1] < 6.636) best\_PVP[14][1] = best\_PVN[14][1] = 0.0;

if (best\_chi\_square[15][1] < 6.636) best\_PVP[15][1] = best\_PVN[15][1] = 0.0;

}

if (p\_value\_to\_progress == .001)

{

if (best\_chi\_square[1][1] < 10.829) best\_PVP[1][1] = best\_PVN[1][1] = 0.0;

if (best\_chi\_square[2][1] < 10.829) best\_PVP[2][1] = best\_PVN[2][1] = 0.0;

if (best\_chi\_square[3][1] < 10.829) best\_PVP[3][1] = best\_PVN[3][1] = 0.0;

if (best\_chi\_square[4][1] < 10.829) best\_PVP[4][1] = best\_PVN[4][1] = 0.0;

if (best\_chi\_square[5][1] < 10.829) best\_PVP[5][1] = best\_PVN[5][1] = 0.0;

if (best\_chi\_square[6][1] < 10.829) best\_PVP[6][1] = best\_PVN[6][1] = 0.0;

if (best\_chi\_square[7][1] < 10.829) best\_PVP[7][1] = best\_PVN[7][1] = 0.0;

if (best\_chi\_square[8][1] < 10.829) best\_PVP[8][1] = best\_PVN[8][1] = 0.0;

if (best\_chi\_square[9][1] < 10.829) best\_PVP[9][1] = best\_PVN[9][1] = 0.0;

if (best\_chi\_square[10][1] < 10.829) best\_PVP[10][1] = best\_PVN[10][1] = 0.0;

if (best\_chi\_square[11][1] < 10.829) best\_PVP[11][1] = best\_PVN[11][1] = 0.0;

if (best\_chi\_square[12][1] < 10.829) best\_PVP[12][1] = best\_PVN[12][1] = 0.0;

if (best\_chi\_square[13][1] < 10.829) best\_PVP[13][1] = best\_PVN[13][1] = 0.0;

if (best\_chi\_square[14][1] < 10.829) best\_PVP[14][1] = best\_PVN[14][1] = 0.0;

if (best\_chi\_square[15][1] < 10.829) best\_PVP[15][1] = best\_PVN[15][1] = 0.0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Print Iteration 1 and total data \*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

printf("\*\*\n");

printf("\*\*\n");

printf("\*\* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\*\* | %6d total cases |\n",total\_n);

printf("\*\* | %5.2f%1c ", percent\_positivef, percent\_sign);

print\_variable\_names(variables\_max);

printf(" |\n");

printf("\*\* |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n");

printf("\*\* |\n");

printf("\*\* \_\_\_\_\_\_|\_\_\_\_\_\_\n");

printf("\*\* | ");

print\_variable\_names\_eight((int)best\_variable[1][1]);

printf(" |\n");

if (best\_LT\_GE[1][1] == 1.0) printf("\*\* | GE |\n");

if (best\_LT\_GE[1][1] == 0.0) printf("\*\* | LT |\n");

printf("\*\* |%8.3f |\n",best\_value[1][1]);

printf("\*\* |\_\_\_\_\_\_\_\_\_\_\_|\n");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Print Iterations 2&3 \*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

printf("\*\* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\*\* \_\_\_\_\_\_|\_\_\_\_\_\_ \*1\* \_\_\_\_\_\_|\_\_\_\_\_\_\n");

printf("\*\* | %6d | ",(int)best\_PVN\_denominator[1][1]);

printf("| %6d |\n",(int)(best\_PVP\_denominator[1][1])); /\*Line 1 PVN and PVP \*/

printf("\*\* | ");

print\_variable\_names\_eight((int)best\_variable[1][1]);

printf(" | | ");

print\_variable\_names\_eight((int)best\_variable[1][1]);

printf(" |\n"); /\*Line 2 Variable Name \*/

printf("\*\* | ");

if (best\_LT\_GE[1][1] == 1.0) printf (" LT | ");

if (best\_LT\_GE[1][1] == 1.0) printf ("| GE |\n");

if (best\_LT\_GE[1][1] == 0.0) printf (" GE | ");

if (best\_LT\_GE[1][1] == 0.0) printf ("| LT |\n"); /\*Line 3 LT or GE \*/

printf("\*\* | ");

printf("%8.3f | | %8.3f |\n",best\_value[1][1],best\_value[1][1]); /\*Line 4 Best Values (cutpoints)\*/

printf("\*\* | ");

printf(" %4.1f%1c | | %4.1f%1c |\n",(float)(100\*(1.0-best\_PVN[1][1])),percent\_sign,(float)(100\*best\_PVP[1][1]),percent\_sign);/\*Line 5 Percentage Positive\*/

printf("\*\* | ");

printf(" (%4d) | | (%4d) |\n",(int)((best\_PVN\_denominator[1][1]\*(1.0-best\_PVN[1][1])+0.5)),(int)((best\_PVP\_denominator[1][1]\*best\_PVP[1][1])+0.5));/\*Line 6 Number? Positive\*/

printf("\*\* |\_\_\_\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_\_\_\_|\n");

printf("\*\* \_\_\_\_\_\_|\_\_\_\_\_\_ \_\_\_\_\_\_|\_\_\_\_\_\_\n");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

printf("\*\* | ");

if (best\_chi\_square[3][1] >= chi\_value\_to\_progress) print\_variable\_names\_eight((int)best\_variable[3][1]); else printf(" ");

printf(" | | ");

if (best\_chi\_square[2][1] >= chi\_value\_to\_progress) print\_variable\_names\_eight((int)best\_variable[2][1]); else printf(" ");

//print\_variable\_names\_eight((int)best\_variable[2][1]);

printf(" |\n");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if(best\_chi\_square[3][1] >= chi\_value\_to\_progress)

{

printf("\*\* | ");

if (best\_LT\_GE[3][1] == 1.0) printf (" GE | ");

if (best\_LT\_GE[3][1] == 0.0) printf (" LT | ");

}

else

{

printf("\*\* | ");

printf (" | ");

}

if(best\_chi\_square[2][1] >= chi\_value\_to\_progress)

{

if (best\_LT\_GE[2][1] == 1.0) printf ("| GE |\n");

if (best\_LT\_GE[2][1] == 0.0) printf ("| LT |\n");

}

else

{

//printf("\*\*11 | ");

printf ("| |\n");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

printf("\*\* | ");

if(best\_chi\_square[3][1] >= chi\_value\_to\_progress)

{

printf("%8.3f | | ",best\_value[3][1]);

}

else

{

printf(" | | ");

}

if(best\_chi\_square[2][1] >= chi\_value\_to\_progress)

{

printf("%8.3f |\n",best\_value[2][1]);

}

else

{

printf(" |\n");

}

printf("\*\* |\_\_\_\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_\_\_\_|\n");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Print Iterations 4 5 6 7 \*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

printf("\*\* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\*\* \_\_\_\_\_\_|\_\_\_\_\_\_ \*3\* \_\_\_\_\_\_|\_\_\_\_\_\_ \_\_\_\_\_\_|\_\_\_\_\_\_ \*2\* \_\_\_\_\_\_|\_\_\_\_\_\_\n");

if (DE\_BUG == 1) printf("XXXXXXXXX 3 %7.2f 2 %7.2f\n", best\_PVN[3][1],best\_PVN[2][1]);

if (DE\_BUG == 1) printf("XXXXXXXXX 3 %7.2f 2 %7.2f\n", best\_PVN[3][1],best\_PVN[2][1]);

if (DE\_BUG == 1) printf ("BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB best\_chi\_square[3][1] = %7.3f chi\_value\_to\_progress = %7.3f\n", best\_chi\_square[3][1], chi\_value\_to\_progress); /\*Number 2&3 times through if Chi Square > criteria .05/.01/.001\*/

if (DE\_BUG == 1) printf ("BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB best\_chi\_square[2][1] = %7.3f chi\_value\_to\_progress = %7.3f\n", best\_chi\_square[2][1], chi\_value\_to\_progress); /\*Number 2&3 times through if Chi Square > criteria .05/.01/.001\*/

if (best\_chi\_square[3][1] >= chi\_value\_to\_progress) printf("\*\* | %6d | | %6d |",(int)best\_PVN\_denominator[3][1],(int)(best\_PVP\_denominator[3][1]));

if (best\_chi\_square[3][1] < chi\_value\_to\_progress) printf("\*\* | | | |");

if (best\_chi\_square[2][1] >= chi\_value\_to\_progress) printf(" | %6d | | %6d |\n",(int)best\_PVN\_denominator[2][1],(int)(best\_PVP\_denominator[2][1]));

if (best\_chi\_square[2][1] < chi\_value\_to\_progress) printf(" | | | |\n");

printf("\*\* | ");

if (best\_chi\_square[3][1] >= chi\_value\_to\_progress) print\_variable\_names\_eight((int)best\_variable[3][1]);

if (best\_chi\_square[3][1] < chi\_value\_to\_progress) print\_variable\_names\_eight\_blank((int)best\_variable[3][1]);

printf(" | | ");

if (best\_chi\_square[3][1] >= chi\_value\_to\_progress) print\_variable\_names\_eight((int)best\_variable[3][1]);

if (best\_chi\_square[3][1] < chi\_value\_to\_progress) print\_variable\_names\_eight\_blank((int)best\_variable[3][1]);

printf(" | | ");

if (best\_chi\_square[2][1] >= chi\_value\_to\_progress) print\_variable\_names\_eight((int)best\_variable[2][1]);

if (best\_chi\_square[2][1] < chi\_value\_to\_progress) print\_variable\_names\_eight\_blank((int)best\_variable[2][1]);

printf(" | | ");

if (best\_chi\_square[2][1] >= chi\_value\_to\_progress) print\_variable\_names\_eight((int)best\_variable[2][1]);

if (best\_chi\_square[2][1] < chi\_value\_to\_progress) print\_variable\_names\_eight\_blank((int)best\_variable[2][1]);

printf(" |\n");

printf("\*\* | ");

if (best\_chi\_square[3][1] < chi\_value\_to\_progress) printf (" | | | ");

if (best\_LT\_GE[3][1] == 1.0 && best\_chi\_square[3][1] >= chi\_value\_to\_progress) printf (" LT | | GE | ");

if (best\_LT\_GE[3][1] == 0.0 && best\_chi\_square[3][1] >= chi\_value\_to\_progress) printf (" GE | | LT | ");

if (best\_chi\_square[2][1] < chi\_value\_to\_progress) printf ("| | | |\n");

if (best\_LT\_GE[2][1] == 1.0 && best\_chi\_square[2][1] >= chi\_value\_to\_progress) printf ("| LT | | GE |\n");

if (best\_LT\_GE[2][1] == 0.0 && best\_chi\_square[2][1] >= chi\_value\_to\_progress) printf ("| GE | | LT |\n");

printf("\*\* ");

if (best\_chi\_square[3][1] >= chi\_value\_to\_progress) printf("| %8.3f | | %8.3f | ",best\_value[3][1],best\_value[3][1]);

if (best\_chi\_square[3][1] < chi\_value\_to\_progress) printf("| | | | ");

if (best\_chi\_square[2][1] >= chi\_value\_to\_progress) printf("| %8.3f | | %8.3f |\n",best\_value[2][1],best\_value[2][1]);

if (best\_chi\_square[2][1] < chi\_value\_to\_progress) printf("| | | |\n");

printf("\*\* ");

if (best\_chi\_square[3][1] >= chi\_value\_to\_progress) printf("| %4.1f%1c | | %4.1f%1c | ",(float)(100\*(1.0-best\_PVN[3][1])),percent\_sign,(float)(100\*best\_PVP[3][1]),percent\_sign);

if (best\_chi\_square[3][1] < chi\_value\_to\_progress) printf("| | | | ");

if (best\_chi\_square[2][1] >= chi\_value\_to\_progress) printf("| %4.1f%1c | | %4.1f%1c |\n",(float)(100\*(1.0-best\_PVN[2][1])),percent\_sign,(float)(100\*best\_PVP[2][1]),percent\_sign);

if (best\_chi\_square[2][1] < chi\_value\_to\_progress) printf("| | | |\n");

printf("\*\* ");

if (best\_chi\_square[3][1] >= chi\_value\_to\_progress) printf("| (%4d) | | (%4d) | ",(int)((best\_PVN\_denominator[3][1]\*(1.0-best\_PVN[3][1])+0.5)),(int)((best\_PVP\_denominator[3][1]\*best\_PVP[3][1])+0.5));

if (best\_chi\_square[3][1] < chi\_value\_to\_progress) printf("| | | | ",(int)((best\_PVN\_denominator[3][1]\*(1.0-best\_PVN[3][1])+0.5)),(int)((best\_PVP\_denominator[3][1]\*best\_PVP[3][1])+0.5));

if (best\_chi\_square[2][1] >= chi\_value\_to\_progress) printf("| (%4d) | | (%4d) |\n",(int)((best\_PVN\_denominator[2][1]\*(1.0-best\_PVN[2][1])+0.5)),(int)((best\_PVP\_denominator[2][1]\*best\_PVP[2][1])+0.5));

if (best\_chi\_square[2][1] < chi\_value\_to\_progress) printf("| | | |\n",(int)((best\_PVN\_denominator[2][1]\*(1.0-best\_PVN[2][1])+0.5)),(int)((best\_PVP\_denominator[2][1]\*best\_PVP[2][1])+0.5));

printf("\*\* |\_\_\_\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_\_\_\_|\n");

printf("\*\* \_\_\_\_\_\_|\_\_\_\_\_\_ \_\_\_\_\_\_|\_\_\_\_\_\_ \_\_\_\_\_\_|\_\_\_\_\_\_ \_\_\_\_\_\_|\_\_\_\_\_\_\n");

printf("\*\* | ");

if (best\_chi\_square[7][1] >= chi\_value\_to\_progress) print\_variable\_names\_eight((int)best\_variable[7][1]);

if (best\_chi\_square[7][1] < chi\_value\_to\_progress) print\_variable\_names\_eight\_blank((int)best\_variable[7][1]);

printf(" | | ");

if (best\_chi\_square[6][1] >= chi\_value\_to\_progress) print\_variable\_names\_eight((int)best\_variable[6][1]);

if (best\_chi\_square[6][1] < chi\_value\_to\_progress) print\_variable\_names\_eight\_blank((int)best\_variable[6][1]);

printf(" | | ");

if (best\_chi\_square[5][1] >= chi\_value\_to\_progress) print\_variable\_names\_eight((int)best\_variable[5][1]);

if (best\_chi\_square[5][1] < chi\_value\_to\_progress) print\_variable\_names\_eight\_blank((int)best\_variable[5][1]);

printf(" | | ");

if (best\_chi\_square[4][1] >= chi\_value\_to\_progress) print\_variable\_names\_eight((int)best\_variable[4][1]);

if (best\_chi\_square[4][1] < chi\_value\_to\_progress) print\_variable\_names\_eight\_blank((int)best\_variable[4][1]);

printf(" |\n");

printf("\*\* | ");

if (best\_chi\_square[7][1] >= chi\_value\_to\_progress && best\_LT\_GE[7][1] == 1.0 && best\_PVN[3][1] != 0.0) printf (" GE | ");

if (best\_chi\_square[7][1] >= chi\_value\_to\_progress && best\_LT\_GE[7][1] == 0.0 && best\_PVN[3][1] != 0.0) printf (" LT | ");

if (best\_chi\_square[7][1] < chi\_value\_to\_progress) printf (" | ");

if (best\_chi\_square[6][1] >= chi\_value\_to\_progress && best\_LT\_GE[6][1] == 1.0 && best\_PVN[3][1] != 0.0) printf ("| GE | ");

if (best\_chi\_square[6][1] >= chi\_value\_to\_progress && best\_LT\_GE[6][1] == 0.0 && best\_PVN[3][1] != 0.0) printf ("| LT | ");

if (best\_chi\_square[6][1] < chi\_value\_to\_progress) printf ("| | ");

if (best\_chi\_square[5][1] >= chi\_value\_to\_progress && best\_LT\_GE[5][1] == 1.0 && best\_PVN[2][1] != 0.0) printf ("| GE |");

if (best\_chi\_square[5][1] >= chi\_value\_to\_progress && best\_LT\_GE[5][1] == 0.0 && best\_PVN[2][1] != 0.0) printf ("| LT |");

if (best\_chi\_square[5][1] < chi\_value\_to\_progress) printf ("| |");

if (best\_chi\_square[4][1] >= chi\_value\_to\_progress && best\_LT\_GE[4][1] == 1.0 && best\_PVN[2][1] != 0.0) printf (" | GE |\n");

if (best\_chi\_square[4][1] >= chi\_value\_to\_progress && best\_LT\_GE[4][1] == 0.0 && best\_PVN[2][1] != 0.0) printf (" | LT |\n");

if (best\_chi\_square[4][1] < chi\_value\_to\_progress) printf (" | |\n");

printf("\*\* ");

if (best\_chi\_square[7][1] >= chi\_value\_to\_progress) printf("| %8.3f | ",best\_value[7][1]);

if (best\_chi\_square[7][1] < chi\_value\_to\_progress) printf("| | ");

if (best\_chi\_square[6][1] >= chi\_value\_to\_progress) printf("| %8.3f | ",best\_value[6][1]);

if (best\_chi\_square[6][1] < chi\_value\_to\_progress) printf("| | ");

if (best\_chi\_square[5][1] >= chi\_value\_to\_progress) printf("| %8.3f | ",best\_value[5][1]);

if (best\_chi\_square[5][1] < chi\_value\_to\_progress) printf("| | ");

if (best\_chi\_square[4][1] >= chi\_value\_to\_progress) printf("| %8.3f |\n",best\_value[4][1]);

if (best\_chi\_square[4][1] < chi\_value\_to\_progress) printf("| |\n");

printf("\*\* |\_\_\_\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_\_\_\_|\n");

if (DE\_BUG == 1) {

printf("XXXXXXXXX 7 %7.2f 6 %7.2f 5 %7.2f 4 %7.2f\n", best\_PVN[7][1],best\_PVN[6][1],best\_PVN[5][1],best\_PVN[4][1] );

printf("XXXXXXXXX 7 %7.2f 6 %7.2f 5 %7.2f 4 %7.2f\n", best\_PVN[7][1],best\_PVN[6][1],best\_PVN[5][1],best\_PVN[4][1] );

printf ("BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB best\_chi\_square[7][1] = %7.3f chi\_value\_to\_progress = %7.3f\n", best\_chi\_square[7][1], chi\_value\_to\_progress); /\*Number 2&3 times through if Chi Square > criteria .05/.01/.001\*/

printf ("BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB best\_chi\_square[6][1] = %7.3f chi\_value\_to\_progress = %7.3f\n", best\_chi\_square[6][1], chi\_value\_to\_progress); /\*Number 2&3 times through if Chi Square > criteria .05/.01/.001\*/

printf ("BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB best\_chi\_square[5][1] = %7.3f chi\_value\_to\_progress = %7.3f\n", best\_chi\_square[5][1], chi\_value\_to\_progress); /\*Number 2&3 times through if Chi Square > criteria .05/.01/.001\*/

printf ("BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB best\_chi\_square[4][1] = %7.3f chi\_value\_to\_progress = %7.3f\n", best\_chi\_square[4][1], chi\_value\_to\_progress); /\*Number 2&3 times through if Chi Square > criteria .05/.01/.001\*/

printf("\n");

printf("XXXXXXXXX 15 %7.2f 14 %7.2f 13 %7.2f 12 %7.2f 11 %7.2f 10 %7.2f 9 %7.2f 8 %7.2f\n", best\_PVN[15][1],best\_PVN[14][1],best\_PVN[13][1],best\_PVN[12][1], best\_PVN[11][1],best\_PVN[10][1],best\_PVN[9][1],best\_PVN[8][1] );

printf("XXXXXXXXX 15 %7.2f 14 %7.2f 13 %7.2f 12 %7.2f 11 %7.2f 10 %7.2f 9 %7.2f 8 %7.2f\n", best\_PVN[15][1],best\_PVN[14][1],best\_PVN[13][1],best\_PVN[12][1], best\_PVN[11][1],best\_PVN[10][1],best\_PVN[9][1],best\_PVN[8][1] );

printf ("BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB best\_chi\_square[7][1] = %7.3f chi\_value\_to\_progress = %7.3f\n", best\_chi\_square[7][1], chi\_value\_to\_progress); /\*Number 2&3 times through if Chi Square > criteria .05/.01/.001\*/

}

printf("\*\* \_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\*\* \_\_\_\_\_\_|\_\_\_\_\_\_ \*7\* \_\_\_\_\_\_|\_\_\_\_\_\_ \_\_\_\_\_\_|\_\_\_\_\_\_ \*6\* \_\_\_\_\_\_|\_\_\_\_\_\_ \_\_\_\_\_\_|\_\_\_\_\_\_ \*5\* \_\_\_\_\_\_|\_\_\_\_\_\_ \_\_\_\_\_\_|\_\_\_\_\_\_ \*4\* \_\_\_\_\_\_|\_\_\_\_\_\_\n");

if (best\_chi\_square[7][1] >= chi\_value\_to\_progress) printf("\*\* | %6d | | %6d |",(int)best\_PVN\_denominator[7][1],(int)(best\_PVP\_denominator[7][1]));

if (best\_chi\_square[7][1] < chi\_value\_to\_progress) printf("\*\* | | | |");

if (best\_chi\_square[6][1] >= chi\_value\_to\_progress) printf(" | %6d | | %6d |",(int)best\_PVN\_denominator[6][1],(int)(best\_PVP\_denominator[6][1]));

if (best\_chi\_square[6][1] < chi\_value\_to\_progress) printf(" | | | |");

if (best\_chi\_square[5][1] >= chi\_value\_to\_progress) printf(" | %6d | | %6d |",(int)best\_PVN\_denominator[5][1],(int)(best\_PVP\_denominator[5][1]));

if (best\_chi\_square[5][1] < chi\_value\_to\_progress) printf(" | | | |");

if (best\_chi\_square[4][1] >= chi\_value\_to\_progress) printf(" | %6d | | %6d |\n",(int)best\_PVN\_denominator[4][1],(int)(best\_PVP\_denominator[4][1]));

if (best\_chi\_square[4][1] < chi\_value\_to\_progress) printf(" | | | |\n");

printf("\*\* | ");

if (best\_chi\_square[7][1] >= chi\_value\_to\_progress) print\_variable\_names\_eight((int)best\_variable[7][1]);

if (best\_chi\_square[7][1] < chi\_value\_to\_progress) print\_variable\_names\_eight\_blank((int)best\_variable[7][1]);

printf(" | | ");

if (best\_chi\_square[7][1] >= chi\_value\_to\_progress) print\_variable\_names\_eight((int)best\_variable[7][1]);

if (best\_chi\_square[7][1] < chi\_value\_to\_progress) print\_variable\_names\_eight\_blank((int)best\_variable[7][1]);

printf(" | | ");

if (best\_chi\_square[6][1] >= chi\_value\_to\_progress) print\_variable\_names\_eight((int)best\_variable[6][1]);

if (best\_chi\_square[6][1] < chi\_value\_to\_progress) print\_variable\_names\_eight\_blank((int)best\_variable[6][1]);

printf(" | | ");

if (best\_chi\_square[6][1] >= chi\_value\_to\_progress) print\_variable\_names\_eight((int)best\_variable[6][1]);

if (best\_chi\_square[6][1] < chi\_value\_to\_progress) print\_variable\_names\_eight\_blank((int)best\_variable[6][1]);

printf(" | | ");

if (best\_chi\_square[5][1] >= chi\_value\_to\_progress) print\_variable\_names\_eight((int)best\_variable[5][1]);

if (best\_chi\_square[5][1] < chi\_value\_to\_progress) print\_variable\_names\_eight\_blank((int)best\_variable[5][1]);

printf(" | | ");

if (best\_chi\_square[5][1] >= chi\_value\_to\_progress) print\_variable\_names\_eight((int)best\_variable[5][1]);

if (best\_chi\_square[5][1] < chi\_value\_to\_progress) print\_variable\_names\_eight\_blank((int)best\_variable[5][1]);

printf(" | | ");

if (best\_chi\_square[4][1] >= chi\_value\_to\_progress) print\_variable\_names\_eight((int)best\_variable[4][1]);

if (best\_chi\_square[4][1] < chi\_value\_to\_progress) print\_variable\_names\_eight\_blank((int)best\_variable[4][1]);

printf(" | | ");

if (best\_chi\_square[4][1] >= chi\_value\_to\_progress) print\_variable\_names\_eight((int)best\_variable[4][1]);

if (best\_chi\_square[4][1] < chi\_value\_to\_progress) print\_variable\_names\_eight\_blank((int)best\_variable[4][1]);

printf(" |\n");

printf("\*\* | ");

if (best\_LT\_GE[7][1] == 1.0 && best\_chi\_square[7][1] >= chi\_value\_to\_progress) printf (" LT | | GE | ");

if (best\_LT\_GE[7][1] == 0.0 && best\_chi\_square[7][1] >= chi\_value\_to\_progress) printf (" GE | | LT | ");

if (best\_chi\_square[7][1] < chi\_value\_to\_progress) printf (" | | | ");

if (best\_LT\_GE[6][1] == 1.0 && best\_chi\_square[6][1] >= chi\_value\_to\_progress) printf ("| LT | | GE | ");

if (best\_LT\_GE[6][1] == 0.0 && best\_chi\_square[6][1] >= chi\_value\_to\_progress) printf ("| GE | | LT | ");

if (best\_chi\_square[6][1] < chi\_value\_to\_progress) printf ("| | | | ");

if (best\_LT\_GE[5][1] == 1.0 && best\_chi\_square[5][1] >= chi\_value\_to\_progress) printf ("| LT | | GE | ");

if (best\_LT\_GE[5][1] == 0.0 && best\_chi\_square[5][1] >= chi\_value\_to\_progress) printf ("| GE | | LT | ");

if (best\_chi\_square[5][1] < chi\_value\_to\_progress) printf ("| | | | ");

if (best\_LT\_GE[4][1] == 1.0 && best\_chi\_square[4][1] >= chi\_value\_to\_progress) printf ("| LT | | GE |\n");

if (best\_LT\_GE[4][1] == 0.0 && best\_chi\_square[4][1] >= chi\_value\_to\_progress) printf ("| GE | | LT |\n");

if (best\_chi\_square[4][1] < chi\_value\_to\_progress) printf ("| | | |\n");

printf("\*\* |");

if (best\_chi\_square[7][1] >= chi\_value\_to\_progress) printf(" %8.3f | | %8.3f | ",best\_value[7][1],best\_value[7][1]);

if (best\_chi\_square[7][1] < chi\_value\_to\_progress) printf(" | | | ");

if (best\_chi\_square[6][1] >= chi\_value\_to\_progress) printf("| %8.3f | | %8.3f | ",best\_value[6][1],best\_value[6][1]);

if (best\_chi\_square[6][1] < chi\_value\_to\_progress) printf("| | | | ");

if (best\_chi\_square[5][1] >= chi\_value\_to\_progress) printf("| %8.3f | | %8.3f | ",best\_value[5][1],best\_value[5][1]);

if (best\_chi\_square[5][1] < chi\_value\_to\_progress) printf("| | | | ");

if (best\_chi\_square[4][1] >= chi\_value\_to\_progress) printf("| %8.3f | | %8.3f |\n",best\_value[4][1],best\_value[4][1]);

if (best\_chi\_square[4][1] < chi\_value\_to\_progress) printf("| | | |\n");

printf("\*\* |");

if (best\_chi\_square[7][1] >= chi\_value\_to\_progress) printf(" %4.1f%1c | | %4.1f%1c | ",(float)(100\*(1.0-best\_PVN[7][1])),percent\_sign,(float)(100\*best\_PVP[7][1]),percent\_sign);

if (best\_chi\_square[7][1] < chi\_value\_to\_progress) printf(" | | | ");

if (best\_chi\_square[6][1] >= chi\_value\_to\_progress) printf("| %4.1f%1c | | %4.1f%1c | ",(float)(100\*(1.0-best\_PVN[6][1])),percent\_sign,(float)(100\*best\_PVP[6][1]),percent\_sign);

if (best\_chi\_square[6][1] < chi\_value\_to\_progress) printf("| | | | ");

if (best\_chi\_square[5][1] >= chi\_value\_to\_progress) printf("| %4.1f%1c | | %4.1f%1c | ",(float)(100\*(1.0-best\_PVN[5][1])),percent\_sign,(float)(100\*best\_PVP[5][1]),percent\_sign);

if (best\_chi\_square[5][1] < chi\_value\_to\_progress) printf("| | | | ");

if (best\_chi\_square[4][1] >= chi\_value\_to\_progress) printf("| %4.1f%1c | | %4.1f%1c |\n",(float)(100\*(1.0-best\_PVN[4][1])),percent\_sign,(float)(100\*best\_PVP[4][1]),percent\_sign);

if (best\_chi\_square[4][1] < chi\_value\_to\_progress) printf("| | | |\n");

printf("\*\* |");

if (best\_chi\_square[7][1] >= chi\_value\_to\_progress) printf(" (%4d) | | (%4d) | ",(int)((best\_PVN\_denominator[7][1]\*(1.0-best\_PVN[7][1])+0.5)),(int)((best\_PVP\_denominator[7][1]\*best\_PVP[7][1])+0.5));

if (best\_chi\_square[7][1] < chi\_value\_to\_progress) printf(" | | | ");

if (best\_chi\_square[6][1] >= chi\_value\_to\_progress) printf("| (%4d) | | (%4d) | ",(int)((best\_PVN\_denominator[6][1]\*(1.0-best\_PVN[6][1])+0.5)),(int)((best\_PVP\_denominator[6][1]\*best\_PVP[6][1])+0.5));

if (best\_chi\_square[6][1] < chi\_value\_to\_progress) printf("| | | | ");

if (best\_chi\_square[5][1] >= chi\_value\_to\_progress) printf("| (%4d) | | (%4d) | ",(int)((best\_PVN\_denominator[5][1]\*(1.0-best\_PVN[5][1])+0.5)),(int)((best\_PVP\_denominator[5][1]\*best\_PVP[5][1])+0.5));

if (best\_chi\_square[5][1] < chi\_value\_to\_progress) printf("| | | | ");

if (best\_chi\_square[4][1] >= chi\_value\_to\_progress) printf("| (%4d) | | (%4d) |\n",(int)((best\_PVN\_denominator[4][1]\*(1.0-best\_PVN[4][1])+0.5)),(int)((best\_PVP\_denominator[4][1]\*best\_PVP[4][1])+0.5));

if (best\_chi\_square[4][1] < chi\_value\_to\_progress) printf("| | | |\n");

printf("\*\* |\_\_\_\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_\_\_\_|\n");

printf("\*\* \_\_\_\_\_\_|\_\_\_\_\_\_ \_\_\_\_\_\_|\_\_\_\_\_\_ \_\_\_\_\_\_|\_\_\_\_\_\_ \_\_\_\_\_\_|\_\_\_\_\_\_ \_\_\_\_\_\_|\_\_\_\_\_\_ \_\_\_\_\_\_|\_\_\_\_\_\_ \_\_\_\_\_\_|\_\_\_\_\_\_ \_\_\_\_\_\_|\_\_\_\_\_\_\n");

printf("\*\* | ");

if (best\_chi\_square[15][1] >= chi\_value\_to\_progress) print\_variable\_names\_eight((int)best\_variable[15][1]);

if (best\_chi\_square[15][1] < chi\_value\_to\_progress) print\_variable\_names\_eight\_blank((int)best\_variable[15][1]);

printf(" | | ");

if (best\_chi\_square[14][1] >= chi\_value\_to\_progress) print\_variable\_names\_eight((int)best\_variable[14][1]);

if (best\_chi\_square[14][1] < chi\_value\_to\_progress) print\_variable\_names\_eight\_blank((int)best\_variable[14][1]);

printf(" | | ");

if (best\_chi\_square[13][1] >= chi\_value\_to\_progress) print\_variable\_names\_eight((int)best\_variable[13][1]);

if (best\_chi\_square[13][1] < chi\_value\_to\_progress) print\_variable\_names\_eight\_blank((int)best\_variable[13][1]);

printf(" | | ");

if (best\_chi\_square[12][1] >= chi\_value\_to\_progress) print\_variable\_names\_eight((int)best\_variable[12][1]);

if (best\_chi\_square[12][1] < chi\_value\_to\_progress) print\_variable\_names\_eight\_blank((int)best\_variable[12][1]);

printf(" | | ");

if (best\_chi\_square[11][1] >= chi\_value\_to\_progress) print\_variable\_names\_eight((int)best\_variable[11][1]);

if (best\_chi\_square[11][1] < chi\_value\_to\_progress) print\_variable\_names\_eight\_blank((int)best\_variable[11][1]);

printf(" | | ");

if (best\_chi\_square[10][1] >= chi\_value\_to\_progress) print\_variable\_names\_eight((int)best\_variable[10][1]);

if (best\_chi\_square[10][1] < chi\_value\_to\_progress) print\_variable\_names\_eight\_blank((int)best\_variable[10][1]);

printf(" | | ");

if (best\_chi\_square[9][1] >= chi\_value\_to\_progress) print\_variable\_names\_eight((int)best\_variable[9][1]);

if (best\_chi\_square[9][1] < chi\_value\_to\_progress) print\_variable\_names\_eight\_blank((int)best\_variable[9][1]);

printf(" | | ");

if (best\_chi\_square[8][1] >= chi\_value\_to\_progress) print\_variable\_names\_eight((int)best\_variable[8][1]);

if (best\_chi\_square[8][1] < chi\_value\_to\_progress) print\_variable\_names\_eight\_blank((int)best\_variable[8][1]);

printf(" |\n");

printf("\*\* | ");

if (best\_LT\_GE[15][1] == 1.0 && best\_chi\_square[15][1] >= chi\_value\_to\_progress) printf (" GE | |");

if (best\_LT\_GE[15][1] == 0.0 && best\_chi\_square[15][1] >= chi\_value\_to\_progress) printf (" LT | |");

if (best\_chi\_square[15][1] < chi\_value\_to\_progress) printf (" | |");

if (best\_LT\_GE[14][1] == 1.0 && best\_chi\_square[14][1] >= chi\_value\_to\_progress) printf (" GE | ");

if (best\_LT\_GE[14][1] == 0.0 && best\_chi\_square[14][1] >= chi\_value\_to\_progress) printf (" LT | ");

if (best\_chi\_square[14][1] < chi\_value\_to\_progress) printf (" | ");

if (best\_LT\_GE[13][1] == 1.0 && best\_chi\_square[13][1] >= chi\_value\_to\_progress) printf ("| GE | ");

if (best\_LT\_GE[13][1] == 0.0 && best\_chi\_square[13][1] >= chi\_value\_to\_progress) printf ("| LT | ");

if (best\_chi\_square[13][1] < chi\_value\_to\_progress) printf ("| | ");

if (best\_LT\_GE[12][1] == 1.0 && best\_chi\_square[12][1] >= chi\_value\_to\_progress) printf ("| GE | ");

if (best\_LT\_GE[12][1] == 0.0 && best\_chi\_square[12][1] >= chi\_value\_to\_progress) printf ("| LT | ");

if (best\_chi\_square[12][1] < chi\_value\_to\_progress) printf ("| | ");

if (best\_LT\_GE[11][1] == 1.0 && best\_chi\_square[11][1] >= chi\_value\_to\_progress) printf ("| GE |");

if (best\_LT\_GE[11][1] == 0.0 && best\_chi\_square[11][1] >= chi\_value\_to\_progress) printf ("| LT |");

if (best\_chi\_square[11][1] < chi\_value\_to\_progress) printf ("| |");

if (best\_LT\_GE[10][1] == 1.0 && best\_chi\_square[10][1] >= chi\_value\_to\_progress) printf (" | GE | ");

if (best\_LT\_GE[10][1] == 0.0 && best\_chi\_square[10][1] >= chi\_value\_to\_progress) printf (" | LT | ");

if (best\_chi\_square[10][1] < chi\_value\_to\_progress) printf (" | | ");

if (best\_LT\_GE[9][1] == 1.0 && best\_chi\_square[9][1] >= chi\_value\_to\_progress) printf ("| GE |");

if (best\_LT\_GE[9][1] == 0.0 && best\_chi\_square[9][1] >= chi\_value\_to\_progress) printf ("| LT |");

if (best\_chi\_square[9][1] < chi\_value\_to\_progress) printf ("| |");

if (best\_LT\_GE[8][1] == 1.0 && best\_chi\_square[8][1] >= chi\_value\_to\_progress) printf (" | GE |\n");

if (best\_LT\_GE[8][1] == 0.0 && best\_chi\_square[8][1] >= chi\_value\_to\_progress) printf (" | LT |\n");

if (best\_chi\_square[8][1] < chi\_value\_to\_progress) printf (" | |\n");

printf("\*\* | ");

if (best\_chi\_square[15][1] >= chi\_value\_to\_progress) printf ("%8.3f | |",best\_value[15][1]);

if (best\_chi\_square[15][1] < chi\_value\_to\_progress) printf (" | |");

if (best\_chi\_square[14][1] >= chi\_value\_to\_progress) printf (" %8.3f | |",best\_value[14][1]);

if (best\_chi\_square[14][1] < chi\_value\_to\_progress) printf (" | |");

if (best\_chi\_square[13][1] >= chi\_value\_to\_progress) printf (" %8.3f | |",best\_value[13][1]);

if (best\_chi\_square[13][1] < chi\_value\_to\_progress) printf (" | |");

if (best\_chi\_square[12][1] >= chi\_value\_to\_progress) printf (" %8.3f | |",best\_value[12][1]);

if (best\_chi\_square[12][1] < chi\_value\_to\_progress) printf (" | |");

if (best\_chi\_square[11][1] >= chi\_value\_to\_progress) printf (" %8.3f |",best\_value[11][1]);

if (best\_chi\_square[11][1] < chi\_value\_to\_progress) printf (" |");

if (best\_chi\_square[10][1] >= chi\_value\_to\_progress) printf (" | %8.3f | ",best\_value[10][1]);

if (best\_chi\_square[10][1] < chi\_value\_to\_progress) printf (" | | ");

if (best\_chi\_square[9][1] >= chi\_value\_to\_progress) printf ("| %8.3f |",best\_value[9][1]);

if (best\_chi\_square[9][1] < chi\_value\_to\_progress) printf ("| |");

if (best\_chi\_square[8][1] >= chi\_value\_to\_progress) printf (" | %8.3f |\n",best\_value[8][1]);

if (best\_chi\_square[8][1] < chi\_value\_to\_progress) printf (" | |\n");

printf("\*\* |\_\_\_\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_\_\_\_|\n");

printf("\*\* \*15\* \*14\* \*13\* \*12\* \*11\* \*10\* \*9\* \*8\*\n");

}