

Planbureau voor de Leefomgeving
Hoe open is de Waddenzee? Een indicator voor de openheid van het Waddenlandschap
F.G. Wortelboer
PBL-publicatienummer 500180001
December 2009

Bijlage 2. Programma voor de ruimtelijke analyse van de openheid van de Waddenzee.

Onderstaand programma is gecompileerd met Microsoft Visual Studio 2008 C++ Express Edition.

Het programma leest ascii-grid-bestanden waarvan wordt aangenomen dat deze dezelfde ruimtelijke eenheid beschrijven (in x- en y-richting en qua grootte van de gridcellen). Voor zichtpunt 1 is als 'pointname' gebruikt: 'wzvp1' (zie ook Bijlage 1). Het programma verwacht:

- een hoogtebestand (oftewel barrierebestand, bijvoorbeeld 'wzvp1_h', zoals aangemaakt volgens de VBA-code in Bijlage 1);
- een bijbehorend code-bestand met codes die het landgebruik of het type van menselijke objecten aangeven (eveneens aangemaakt volgens de VBA-code in Bijlage 1);
- en een bestand dat het resultaat is van de viewshed-analyse in Arc/Info (met 1-en voor plaatsen die zichtbaar zijn vanuit het zichtpunt en 0-en voor de plaatsen die niet zichtbaar zijn).

Het onderstaande programma geeft een karakterisering van het zichtbare gebied in termen van het landgebruik en de menselijke objecten. De resultaten worden weggeschreven naar ascii-bestanden die direct gekopieerd kunnen worden naar het spreadsheet waarin de figuren van het rapport gemaakt zijn; zie hiervoor Bijlage 3).

Hieronder volgt de C++-code.

```

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

#include "stdafx.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <malloc.h>
#include <math.h>
#include <ctype.h>

double **LeesAsciiGridDbl(char *FileName);
char **LeesAsciiGridByte(char *FileName);
bool **LeesAsciiGridBool(char *FileName);
int **LeesAsciiGridInt(char *FileName, double ConversionFactor);
short **LeesAsciiGridShort(char *FileName);
long splitstring(char** arrstr, char* strInput);

int **intmatrix(int nrl,int nrh,int ncl,int nch);
void free_intmatrix(int **m, int nrl, int nrh, int ncl, int nch);
double **doublematrix(int nrl,int nrh,int ncl,int nch);
void free_doublematrix(double **m, int nrl, int nrh, int ncl, int nch);
long **longmatrix(int nrl,int nrh,int ncl,int nch);
void free_longmatrix(long **m, int nrl, int nrh, int ncl, int nch);
long ***long3Dmatrix(int nrl,int nrh,int ncl,int nch, int nsl, int nsh);
void free_long3Dmatrix(long ***m, int nrl, int nrh, int ncl, int nch, int nsl, int nsh);
short **shortmatrix(int nrl,int nrh,int ncl,int nch);
void free_shortmatrix(short **m, int nrl, int nrh, int ncl, int nch);
char **charmatrix(int nrl,int nrh,int ncl,int nch);
void free_charmatrix(char **m, int nrl, int nrh, int ncl, int nch);
bool **boolmatrix(int nrl,int nrh,int ncl,int nch);
void free_boolmatrix(bool **m, int nrl, int nrh, int ncl, int nch);

long *longvector(int nrl,int nrh);
double *doublevector(int nrl,int nrh);

```

```

#define MaxLineLength 100000
#define MaxWordsPerLine 20000
#define MISSINGVALUE -99
#define MAXHEIGHT 20000 // 20000 // cm
#define MAXAFSTAND 2500 // decameters
#define MAXCODES 6
#define MAXSECTOR 8
#define MAXCYLINDER 3
#define MAXAREA 83 // 83
#define GRIDSIZEX 10000
#define GRIDSIZEY 10000
#define CELLSIZE 5 // m

int _tmain(int argc, char* argv[])
{
    FILE *fp;
    printf("Analyseren zichtbaarheid per sector per area\n");
    time_t BeginTime, EndTime;
    int i, j, k;
    int iSector;
    int iCylinder;
    int iArea;

    time(&BeginTime);
    printf("Begin programma: tijd=%ld\n",BeginTime);

    int **arr_hint = NULL;
    char **arr_c = NULL;
    bool **arr_vss = NULL;
    char **arr_area = NULL;

//    int *p_hint;
//    char *p_c;
//    bool *p_vss;
//    char *p_area;

    long *count_area; //[(MAXAREA+1)];
    long *count_code; //[(MAXCODES+1)];

```

```

long **count_c; //[MAXAREA+1][MAXCODES+1];
double **max_hint; //[MAXAREA+1][MAXCODES+1];

count_area = longvector(0,MAXAREA+1);
count_code = longvector(0,MAXCODES+1);
count_c = longmatrix(0,MAXAREA+1,0,MAXCODES+1);

long *count_hint[MAXAREA+1][MAXCODES+1];
for(i=0;i<=MAXAREA;i++){
    for(j=0;j<=MAXCODES;j++){
        count_hint[i][j] = new long[MAXHEIGHT+1];
    }
}
max_hint = doublematrix(0,MAXAREA+1,0,MAXCODES+1);

long *aantal_afstand[MAXAREA+1][MAXCODES+1];
for(i=0;i<=MAXAREA;i++){
    for(j=0;j<=MAXCODES;j++){
        aantal_afstand[i][j] = new long[MAXAFSTAND+1];
    }
}

// initialisatie
for(j=0;j<=MAXCODES;j++){
    count_code[j] = 0;
}
for(i=0;i<=MAXAREA;i++){
    count_area[i] = 0;
    for(j=0;j<=MAXCODES;j++){
        count_c[i][j] = 0;
        max_hint[i][j] = -9999;
        for(k=0;k<=MAXHEIGHT;k++){
            count_hint[i][j][k] = 0;
        }
        for(k=0;k<=MAXAFSTAND;k++){
            aantal_afstand[i][j][k] = 0;
        }
    }
}

```

```

    }
}

// statistieken
// Hoogte
long aantalHoogteTotaal;
long iAantalHoogte;
long maxHoogteInt;
double minHoogte, p10Hoogte, p25Hoogte, p50Hoogte, p75Hoogte, p90Hoogte, maxHoogte;
double somHoogte, gemHoogte, stdevHoogte;
// Per Sector
long *iAantalHoogteSector;
long *aantalHoogteSector;
double *minHoogteSector, *p10HoogteSector, *p25HoogteSector, *p50HoogteSector, *p75HoogteSector, *p90HoogteSector, *maxHoogteSector;
double *somHoogteSector, *gemHoogteSector, *stdevHoogteSector;
// Per Cylinder
long *iAantalHoogteCylinder;
long *aantalHoogteCylinder;
double *minHoogteCylinder, *p10HoogteCylinder, *p25HoogteCylinder, *p50HoogteCylinder, *p75HoogteCylinder, *p90HoogteCylinder,
    *maxHoogteCylinder;
double *somHoogteCylinder, *gemHoogteCylinder, *stdevHoogteCylinder;
// Per Area
long *iAantalHoogteArea;
long *aantalHoogteArea;
double *minHoogteArea, *p10HoogteArea, *p25HoogteArea, *p50HoogteArea, *p75HoogteArea, *p90HoogteArea, *maxHoogteArea;
double *somHoogteArea, *gemHoogteArea, *stdevHoogteArea;
// Per Area en Code
long iAantalHoogteAreaCode;
long **aantalHoogteAreaCode;
double **minHoogteAreaCode, **p10HoogteAreaCode, **p25HoogteAreaCode, **p50HoogteAreaCode, **p75HoogteAreaCode, **p90HoogteAreaCode,
    **maxHoogteAreaCode;
double **somHoogteAreaCode, **gemHoogteAreaCode, **stdevHoogteAreaCode;

// Afstand
double Afstand;
long intAfstand;
long aantalAfstandTotaal;
long iAantalAfstand;

```

```

long maxAfstandInt;
double minAfstand, p10Afstand, p25Afstand, p50Afstand, p75Afstand, p90Afstand, maxAfstand;
double somAfstand, gemAfstand, stdevAfstand;
// Per Sector
long *iAantalAfstandSector;
long *aantalAfstandSector;
double *minAfstandSector, *p10AfstandSector, *p25AfstandSector, *p50AfstandSector, *p75AfstandSector, *p90AfstandSector, *maxAfstandSector;
double *somAfstandSector, *gemAfstandSector, *stdevAfstandSector;
// Per Cylinder
long *iAantalAfstandCylinder;
long *aantalAfstandCylinder;
double *minAfstandCylinder, *p10AfstandCylinder, *p25AfstandCylinder, *p50AfstandCylinder, *p75AfstandCylinder, *p90AfstandCylinder,
    *maxAfstandCylinder;
double *somAfstandCylinder, *gemAfstandCylinder, *stdevAfstandCylinder;
// Per Area
long *iAantalAfstandArea;
long *aantalAfstandArea;
double *minAfstandArea, *p10AfstandArea, *p25AfstandArea, *p50AfstandArea, *p75AfstandArea, *p90AfstandArea, *maxAfstandArea;
double *somAfstandArea, *gemAfstandArea, *stdevAfstandArea;
// Per Area en Code
long iAantalAfstandAreaCode;
long **aantalAfstandAreaCode;
double **minAfstandAreaCode, **p10AfstandAreaCode, **p25AfstandAreaCode, **p50AfstandAreaCode, **p75AfstandAreaCode,
    **p90AfstandAreaCode, **maxAfstandAreaCode;
double **somAfstandAreaCode, **gemAfstandAreaCode, **stdevAfstandAreaCode;

// Hoogte
// Sector
iAantalHoogteSector = longvector(0,MAXSECTOR+1);
aantalHoogteSector = longvector(0,MAXSECTOR+1);
minHoogteSector = doublevector(0,MAXSECTOR+1);
p10HoogteSector = doublevector(0,MAXSECTOR+1);
p25HoogteSector = doublevector(0,MAXSECTOR+1);
p50HoogteSector = doublevector(0,MAXSECTOR+1);
p75HoogteSector = doublevector(0,MAXSECTOR+1);
p90HoogteSector = doublevector(0,MAXSECTOR+1);
maxHoogteSector = doublevector(0,MAXSECTOR+1);

```

```

somHoogteSector = doublevector(0,MAXSECTOR+1);
gemHoogteSector = doublevector(0,MAXSECTOR+1);
stdevHoogteSector = doublevector(0,MAXSECTOR+1);
// Cylinder
iAantalHoogteCylinder = longvector(0,MAXCYLINDER+1);
aantalHoogteCylinder = longvector(0,MAXCYLINDER+1);
minHoogteCylinder = doublevector(0,MAXCYLINDER+1);
p10HoogteCylinder = doublevector(0,MAXCYLINDER+1);
p25HoogteCylinder = doublevector(0,MAXCYLINDER+1);
p50HoogteCylinder = doublevector(0,MAXCYLINDER+1);
p75HoogteCylinder = doublevector(0,MAXCYLINDER+1);
p90HoogteCylinder = doublevector(0,MAXCYLINDER+1);
maxHoogteCylinder = doublevector(0,MAXCYLINDER+1);
somHoogteCylinder = doublevector(0,MAXCYLINDER+1);
gemHoogteCylinder = doublevector(0,MAXCYLINDER+1);
stdevHoogteCylinder = doublevector(0,MAXCYLINDER+1);
// Area
iAantalHoogteArea = longvector(0,MAXAREA+1);
aantalHoogteArea = longvector(0,MAXAREA+1);
minHoogteArea = doublevector(0,MAXAREA+1);
p10HoogteArea = doublevector(0,MAXAREA+1);
p25HoogteArea = doublevector(0,MAXAREA+1);
p50HoogteArea = doublevector(0,MAXAREA+1);
p75HoogteArea = doublevector(0,MAXAREA+1);
p90HoogteArea = doublevector(0,MAXAREA+1);
maxHoogteArea = doublevector(0,MAXAREA+1);
somHoogteArea = doublevector(0,MAXAREA+1);
gemHoogteArea = doublevector(0,MAXAREA+1);
stdevHoogteArea = doublevector(0,MAXAREA+1);
// Area en Code
aantalHoogteAreaCode = longmatrix(0,MAXAREA+1,0,MAXCODES+1);
minHoogteAreaCode = doublematrix(0,MAXAREA+1,0,MAXCODES+1);
p10HoogteAreaCode = doublematrix(0,MAXAREA+1,0,MAXCODES+1);
p25HoogteAreaCode = doublematrix(0,MAXAREA+1,0,MAXCODES+1);
p50HoogteAreaCode = doublematrix(0,MAXAREA+1,0,MAXCODES+1);
p75HoogteAreaCode = doublematrix(0,MAXAREA+1,0,MAXCODES+1);
p90HoogteAreaCode = doublematrix(0,MAXAREA+1,0,MAXCODES+1);
maxHoogteAreaCode = doublematrix(0,MAXAREA+1,0,MAXCODES+1);

```

```

somHoogteAreaCode = doublematrix(0,MAXAREA+1,0,MAXCODES+1);
gemHoogteAreaCode = doublematrix(0,MAXAREA+1,0,MAXCODES+1);
stdevHoogteAreaCode = doublematrix(0,MAXAREA+1,0,MAXCODES+1);

// Afstand
// Sector
iAantalAfstandSector = longvector(0,MAXSECTOR+1);
aantalAfstandSector = longvector(0,MAXSECTOR+1);
minAfstandSector = doublevector(0,MAXSECTOR+1);
p10AfstandSector = doublevector(0,MAXSECTOR+1);
p25AfstandSector = doublevector(0,MAXSECTOR+1);
p50AfstandSector = doublevector(0,MAXSECTOR+1);
p75AfstandSector = doublevector(0,MAXSECTOR+1);
p90AfstandSector = doublevector(0,MAXSECTOR+1);
maxAfstandSector = doublevector(0,MAXSECTOR+1);
somAfstandSector = doublevector(0,MAXSECTOR+1);
gemAfstandSector = doublevector(0,MAXSECTOR+1);
stdevAfstandSector = doublevector(0,MAXSECTOR+1);
// Cylinder
iAantalAfstandCylinder = longvector(0,MAXCYLINDER+1);
aantalAfstandCylinder = longvector(0,MAXCYLINDER+1);
minAfstandCylinder = doublevector(0,MAXCYLINDER+1);
p10AfstandCylinder = doublevector(0,MAXCYLINDER+1);
p25AfstandCylinder = doublevector(0,MAXCYLINDER+1);
p50AfstandCylinder = doublevector(0,MAXCYLINDER+1);
p75AfstandCylinder = doublevector(0,MAXCYLINDER+1);
p90AfstandCylinder = doublevector(0,MAXCYLINDER+1);
maxAfstandCylinder = doublevector(0,MAXCYLINDER+1);
somAfstandCylinder = doublevector(0,MAXCYLINDER+1);
gemAfstandCylinder = doublevector(0,MAXCYLINDER+1);
stdevAfstandCylinder = doublevector(0,MAXCYLINDER+1);
// Area
iAantalAfstandArea = longvector(0,MAXAREA+1);
aantalAfstandArea = longvector(0,MAXAREA+1);
minAfstandArea = doublevector(0,MAXAREA+1);
p10AfstandArea = doublevector(0,MAXAREA+1);
p25AfstandArea = doublevector(0,MAXAREA+1);
p50AfstandArea = doublevector(0,MAXAREA+1);

```

```

p75AfstandArea = doublevector(0,MAXAREA+1);
p90AfstandArea = doublevector(0,MAXAREA+1);
maxAfstandArea = doublevector(0,MAXAREA+1);
somAfstandArea = doublevector(0,MAXAREA+1);
gemAfstandArea = doublevector(0,MAXAREA+1);
stdevAfstandArea = doublevector(0,MAXAREA+1);
// Area en Code
aantalAfstandAreaCode = longmatrix(0,MAXAREA+1,0,MAXCODES+1);
minAfstandAreaCode = doublematrix(0,MAXAREA+1,0,MAXCODES+1);
p10AfstandAreaCode = doublematrix(0,MAXAREA+1,0,MAXCODES+1);
p25AfstandAreaCode = doublematrix(0,MAXAREA+1,0,MAXCODES+1);
p50AfstandAreaCode = doublematrix(0,MAXAREA+1,0,MAXCODES+1);
p75AfstandAreaCode = doublematrix(0,MAXAREA+1,0,MAXCODES+1);
p90AfstandAreaCode = doublematrix(0,MAXAREA+1,0,MAXCODES+1);
maxAfstandAreaCode = doublematrix(0,MAXAREA+1,0,MAXCODES+1);
somAfstandAreaCode = doublematrix(0,MAXAREA+1,0,MAXCODES+1);
gemAfstandAreaCode = doublematrix(0,MAXAREA+1,0,MAXCODES+1);
stdevAfstandAreaCode = doublematrix(0,MAXAREA+1,0,MAXCODES+1);

// lees ascii-grids
char basisfilename[100], resfilename[100], filename[100];
char pointname[100];
char c;
printf("Gegevens voor analyse:\nGeef naam van punt: ");
scanf("%s",pointname);
printf("Analyse resultaten punt %s\n", pointname);

strcpy(basisfilename,"G:\\openheid\\resultaten\\");
strcat(basisfilename,pointname);
strcpy(resfilename,basisfilename);
strcat(resfilename,"_res.txt");
strcpy(filename,basisfilename);strcat(filename,"_h.asc");
printf("Inlezen %s_h ...\n",pointname);
arr_hint = LeesAsciiGridInt(filename,100);
printf("Inlezen %s_c ...\n",pointname);
strcpy(filename,basisfilename);strcat(filename,"_c.asc");
arr_c = LeesAsciiGridByte(filename);
printf("Inlezen %s_vss ...\n",pointname);

```

```

strcpy(filename,basisfilename);strcat(filename,"_vss2.asc");
arr_vss = LeesAsciiGridBool(filename);
printf("Inlezen %s_area ...\n",pointname);
strcpy(filename,basisfilename);
c = filename[strlen(filename)-1];
if(!isdigit(c)) {
    filename[strlen(filename)-1] = '\0';
}
strcat(filename,"_area.asc");
arr_area = LeesAsciiGridByte(filename);

// doe analyse
printf("Uitvoeren analyse ...\n");

// Hoogte
aantalHoogteTotaal = 0;
somHoogte = 0.0;
minHoogte = 9999.0;
maxHoogte = -9999.0;
maxHoogteInt = -99;
// Afstand
aantalAfstandTotaal = 0;
somAfstand = 0.0;
minAfstand = 99999.0;
maxAfstand = -9999.0;

for(i=0;i<=MAXSECTOR;i++){
    aantalHoogteSector[i] = 0;
    somHoogteSector[i] = 0.0;
    minHoogteSector[i] = 9999.0;
    maxHoogteSector[i] = -9999.0;
    aantalAfstandSector[i] = 0;
    somAfstandSector[i] = 0.0;
    minAfstandSector[i] = 99999.0;
    maxAfstandSector[i] = -9999.0;
}
for(i=0;i<=MAXCYLINDER;i++){
    aantalHoogteCylinder[i] = 0;

```

```

    somHoogteCylinder[i] = 0.0;
    minHoogteCylinder[i] = 9999.0;
    maxHoogteCylinder[i] = -9999.0;
    aantalAfstandCylinder[i] = 0;
    somAfstandCylinder[i] = 0.0;
    minAfstandCylinder[i] = 99999.0;
    maxAfstandCylinder[i] = -9999.0;
}
for(i=0;i<=MAXAREA;i++){
    aantalHoogteArea[i] = 0;
    somHoogteArea[i] = 0.0;
    minHoogteArea[i] = 9999.0;
    maxHoogteArea[i] = -9999.0;
    aantalAfstandArea[i] = 0;
    somAfstandArea[i] = 0.0;
    minAfstandArea[i] = 99999.0;
    maxAfstandArea[i] = -9999.0;
    for(j=0;j<=MAXCODES;j++){
        // Hoogte
        aantalHoogteAreaCode[i][j] = 0;
        somHoogteAreaCode[i][j] = 0.0;
        minHoogteAreaCode[i][j] = 9999.0;
        maxHoogteAreaCode[i][j] = -9999.0;
        // Afstand
        aantalAfstandAreaCode[i][j] = 0;
        somAfstandAreaCode[i][j] = 0.0;
        minAfstandAreaCode[i][j] = 99999.0;
        maxAfstandAreaCode[i][j] = -9999.0;
    }
}

for(i=0;i<GRIDSIZEX;i++) {
    for (j=0;j<GRIDSIZEY;j++) {
        if(arr_vss[i][j]) {
            if(arr_c[i][j]!=MISSINGVALUE && arr_hint[i][j]!=MISSINGVALUE && arr_area[i][j]!=MISSINGVALUE) {
                if( arr_area[i][j]>10 && (arr_area[i][j] % 10 == 1 || arr_area[i][j] % 10 == 2 || arr_area[i][j] % 10 == 3) &&
                    arr_area[i][j]<=MAXAREA ) {
                    if(arr_c[i][j]>=0 && arr_c[i][j]<=MAXCODES) {

```

```

count_area[arr_area[i][j]]++;
count_code[arr_c[i][j]]++;
count_c[arr_area[i][j]][arr_c[i][j]]++;
if(arr_hint[i][j]<=0) {
    count_hint[arr_area[i][j]][arr_c[i][j]][0]++;
}
else {
    count_hint[arr_area[i][j]][arr_c[i][j]][arr_hint[i][j]]++;
}
// Hoogte
if(arr_hint[i][j]>max_hint[arr_area[i][j]][arr_c[i][j]]) max_hint[arr_area[i][j]][arr_c[i][j]] = arr_hint[i][j];
// Algemeen
aantalHoogteTotaal++;
somHoogte += arr_hint[i][j]/100.0;
if(arr_hint[i][j]>maxHoogteInt) maxHoogteInt = arr_hint[i][j];
if(arr_hint[i][j]/100.0 < minHoogte) minHoogte = arr_hint[i][j] / 100.0;
if(arr_hint[i][j]/100.0 > maxHoogte) maxHoogte = arr_hint[i][j] / 100.0;
// Sector
iSector = (arr_area[i][j] - (arr_area[i][j] % 10))/10;
aantalHoogteSector[iSector]++;
if(arr_hint[i][j]/100.0 < minHoogteSector[iSector]) minHoogteSector[iSector] = arr_hint[i][j]/100.0;
if(arr_hint[i][j]/100.0 > maxHoogteSector[iSector]) maxHoogteSector[iSector] = arr_hint[i][j]/100.0;
somHoogteSector[iSector] += arr_hint[i][j]/100.0;
// Cylinder
iCylinder = arr_area[i][j] % 10;
aantalHoogteCylinder[iCylinder]++;
if(arr_hint[i][j]/100.0 < minHoogteCylinder[iCylinder])
    minHoogteCylinder[iCylinder] = arr_hint[i][j]/100.0;
if(arr_hint[i][j]/100.0 > maxHoogteCylinder[iCylinder])
    maxHoogteCylinder[iCylinder] = arr_hint[i][j]/100.0;
somHoogteCylinder[iCylinder] += arr_hint[i][j]/100.0;
// Area
aantalHoogteArea[arr_area[i][j]]++;
if(arr_hint[i][j]/100.0 < minHoogteArea[arr_area[i][j]])
    minHoogteArea[arr_area[i][j]] = arr_hint[i][j]/100.0;
if(arr_hint[i][j]/100.0 > maxHoogteArea[arr_area[i][j]])
    maxHoogteArea[arr_area[i][j]] = arr_hint[i][j]/100.0;
somHoogteArea[arr_area[i][j]] += arr_hint[i][j]/100.0;

```

```

// Area en Code
aantalHoogteAreaCode[arr_area[i][j]][arr_c[i][j]]++;
if(arr_hint[i][j]/100.0 < minHoogteAreaCode[arr_area[i][j]][arr_c[i][j]])
    minHoogteAreaCode[arr_area[i][j]][arr_c[i][j]] = arr_hint[i][j]/100.0;
if(arr_hint[i][j]/100.0 > maxHoogteAreaCode[arr_area[i][j]][arr_c[i][j]])
    maxHoogteAreaCode[arr_area[i][j]][arr_c[i][j]] = arr_hint[i][j]/100.0;
somHoogteAreaCode[arr_area[i][j]][arr_c[i][j]] += arr_hint[i][j]/100.0;

// Afstand
Afstand = sqrt( pow( abs(i-GRIDSIZEX/2)*CELLSIZE,2.0 ) + pow(abs(j-GRIDSIZEY/2)*CELLSIZE,2.0)
);

intAfstand = (long)(Afstand/10); // intAfstand in decameters
if(intAfstand>=0 && intAfstand<=MAXAFSTAND) {
    aantal_afstand[arr_area[i][j]][arr_c[i][j]][intAfstand]++;
}
else {
    printf("Fout in afstand: i=%d j=%d intAfstand=%d\n", arr_area[i][j], arr_c[i][j], intAfstand);
    getchar();
}
// Algemeen
aantalAfstandTotaal++;
somAfstand += Afstand;
if(Afstand < minAfstand) minAfstand = Afstand;
if(Afstand > maxAfstand) maxAfstand = Afstand;
// Sector
iSector = (arr_area[i][j] - (arr_area[i][j] % 10))/10;
aantalAfstandSector[iSector]++;
if(Afstand < minAfstandSector[iSector]) minAfstandSector[iSector] = Afstand;
if(Afstand > maxAfstandSector[iSector]) maxAfstandSector[iSector] = Afstand;
somAfstandSector[iSector] += Afstand;
// Cylinder
iCylinder = arr_area[i][j] % 10;
aantalAfstandCylinder[iCylinder]++;
if(Afstand < minAfstandCylinder[iCylinder]) minAfstandCylinder[iCylinder] = Afstand;
if(Afstand > maxAfstandCylinder[iCylinder]) maxAfstandCylinder[iCylinder] = Afstand;
somAfstandCylinder[iCylinder] += Afstand;
// Area
aantalAfstandArea[arr_area[i][j]]++;

```



```

else {
    gemAfstandSector[iSector] = -99;
}
stdevAfstandSector[iSector] = 0.0;
}
for(iCylinder=0;iCylinder<=MAXCYLINDER;iCylinder++){
    if(aantalHoogteCylinder[iCylinder] > 0) {
        gemHoogteCylinder[iCylinder] = somHoogteCylinder[iCylinder] / aantalHoogteCylinder[iCylinder];
    }
    else {
        gemHoogteCylinder[iCylinder] = -99;
    }
    stdevHoogteCylinder[iCylinder] = 0.0;
    if(aantalAfstandCylinder[iCylinder] > 0) {
        gemAfstandCylinder[iCylinder] = somAfstandCylinder[iCylinder] / aantalAfstandCylinder[iCylinder];
    }
    else {
        gemAfstandCylinder[iCylinder] = -99;
    }
    stdevAfstandCylinder[iCylinder] = 0.0;
}
for(i=0;i<=MAXAREA;i++){
    iSector = (i - (i % 10))/10;
    iCylinder = i % 10;
    if( i>10 && (i % 10 == 1 || i % 10 == 2 || i % 10 == 3) ) {
        if(aantalHoogteArea[i] > 0) {
            gemHoogteArea[i] = somHoogteArea[i] / aantalHoogteArea[i];
        }
        else {
            gemHoogteArea[i] = -99;
        }
        stdevHoogteArea[i] = 0.0;
        if(aantalAfstandArea[i] > 0) {
            gemAfstandArea[i] = somAfstandArea[i] / aantalAfstandArea[i];
        }
        else {
            gemAfstandArea[i] = -99;
        }
    }
}

```

```

stdevAfstandArea[i] = 0.0;
for(j=0;j<=MAXCODES;j++){
    // Hoogte
    if(aantalHoogteAreaCode[i][j] > 0) {
        gemHoogteAreaCode[i][j] = somHoogteAreaCode[i][j] / aantalHoogteAreaCode[i][j];
    }
    else {
        gemHoogteAreaCode[i][j] = -99;
    }
    iAantalHoogteAreaCode = 0;
    p10HoogteAreaCode[i][j] = -999.0;
    p25HoogteAreaCode[i][j] = -999.0;
    p50HoogteAreaCode[i][j] = -999.0;
    p75HoogteAreaCode[i][j] = -999.0;
    p90HoogteAreaCode[i][j] = -999.0;
    stdevHoogteAreaCode[i][j] = 0.0;
    for(k=0;k<=MAXHEIGHT;k++){
        if(count_hint[i][j][k]>0) {
            // Algemeen
            stdevHoogte += count_hint[i][j][k] * pow((gemHoogte - k/100.0),2.0);
            // Sector
            stdevHoogteSector[iSector] += count_hint[i][j][k] * pow((gemHoogteSector[iSector] - k/100.0),2.0);
            // Cylinder
            stdevHoogteCylinder[iCylinder] += count_hint[i][j][k] * pow((gemHoogteCylinder[iCylinder] - k/100.0),2.0);
            // Area
            stdevHoogteArea[i] += count_hint[i][j][k] * pow((gemHoogteArea[i] - k/100.0),2.0);
            // Area en Code
            iAantalHoogteAreaCode += count_hint[i][j][k];
            if(iAantalHoogteAreaCode>0.1*aantalHoogteAreaCode[i][j] && p10HoogteAreaCode[i][j]<-99.0)
                p10HoogteAreaCode[i][j] = k/100.0;
            if(iAantalHoogteAreaCode>0.25*aantalHoogteAreaCode[i][j] && p25HoogteAreaCode[i][j]<-99.0)
                p25HoogteAreaCode[i][j] = k/100.0;
            if(iAantalHoogteAreaCode>0.5*aantalHoogteAreaCode[i][j] && p50HoogteAreaCode[i][j]<-99.0)
                p50HoogteAreaCode[i][j] = k/100.0;
            if(iAantalHoogteAreaCode>0.75*aantalHoogteAreaCode[i][j] && p75HoogteAreaCode[i][j]<-99.0)
                p75HoogteAreaCode[i][j] = k/100.0;
            if(iAantalHoogteAreaCode>0.9*aantalHoogteAreaCode[i][j] && p90HoogteAreaCode[i][j]<-99.0)
                p90HoogteAreaCode[i][j] = k/100.0;
        }
    }
}

```

```

        stdevHoogteAreaCode[i][j] += count_hint[i][j][k] * pow((gemHoogteAreaCode[i][j] - k/100.0),2.0);
    }
}
// Afstand
if(aantalAfstandAreaCode[i][j] > 0) {
    gemAfstandAreaCode[i][j] = somAfstandAreaCode[i][j] / aantalAfstandAreaCode[i][j];
}
else {
    gemAfstandAreaCode[i][j] = -99;
}
iAantalAfstandAreaCode = 0;
p10AfstandAreaCode[i][j] = -999.0;
p25AfstandAreaCode[i][j] = -999.0;
p50AfstandAreaCode[i][j] = -999.0;
p75AfstandAreaCode[i][j] = -999.0;
p90AfstandAreaCode[i][j] = -999.0;
stdevAfstandAreaCode[i][j] = 0.0;
for(k=0;k<=MAXAFSTAND;k++){
    if(aantal_afstand[i][j][k]>0) {
        // Algemeen
        stdevAfstand += aantal_afstand[i][j][k] * pow((gemAfstand - k*10.0),2.0);
        // Sector
        stdevAfstandSector[iSector] += aantal_afstand[i][j][k] * pow((gemAfstandSector[iSector] - k*10.0),2.0);
        // Cylinder
        stdevAfstandCylinder[iCylinder] += aantal_afstand[i][j][k] * pow((gemAfstandCylinder[iCylinder] - k*10.0),2.0);
        // Area
        stdevAfstandArea[i] += aantal_afstand[i][j][k] * pow((gemAfstandArea[i] - k*10.0),2.0);
        // Area en Code
        iAantalAfstandAreaCode += aantal_afstand[i][j][k];
        if(iAantalAfstandAreaCode>0.1*aantalAfstandAreaCode[i][j] && p10AfstandAreaCode[i][j]<-99.0)
            p10AfstandAreaCode[i][j] = k*10.0;
        if(iAantalAfstandAreaCode>0.25*aantalAfstandAreaCode[i][j] && p25AfstandAreaCode[i][j]<-99.0)
            p25AfstandAreaCode[i][j] = k*10.0;
        if(iAantalAfstandAreaCode>0.5*aantalAfstandAreaCode[i][j] && p50AfstandAreaCode[i][j]<-99.0)
            p50AfstandAreaCode[i][j] = k*10.0;
        if(iAantalAfstandAreaCode>0.75*aantalAfstandAreaCode[i][j] && p75AfstandAreaCode[i][j]<-99.0)
            p75AfstandAreaCode[i][j] = k*10.0;
        if(iAantalAfstandAreaCode>0.9*aantalAfstandAreaCode[i][j] && p90AfstandAreaCode[i][j]<-99.0)

```

```

        p90AfstandAreaCode[i][j] = k*10.0;
        stdevAfstandAreaCode[i][j] += aantal_afstand[i][j][k] * pow((gemAfstandAreaCode[i][j] - k*10.0),2.0);
    }
}
    stdevHoogteAreaCode[i][j] = pow(stdevHoogteAreaCode[i][j] / (aantalHoogteAreaCode[i][j] - 1),0.5);
    stdevAfstandAreaCode[i][j] = pow(stdevAfstandAreaCode[i][j] / (aantalAfstandAreaCode[i][j] - 1),0.5);
}
    stdevHoogteArea[i] = pow(stdevHoogteArea[i] / (aantalHoogteArea[i] - 1),0.5);
    stdevAfstandArea[i] = pow(stdevAfstandArea[i] / (aantalAfstandArea[i] - 1),0.5);
}
}
stdevHoogte = pow(stdevHoogte / (aantalHoogteTotaal - 1),0.5);
stdevAfstand = pow(stdevAfstand / (aantalAfstandTotaal - 1),0.5);
for(iSector=0;iSector<=MAXSECTOR;iSector++){
    stdevHoogteSector[iSector] = pow(stdevHoogteSector[iSector] / (aantalHoogteSector[iSector] - 1),0.5);
    stdevAfstandSector[iSector] = pow(stdevAfstandSector[iSector] / (aantalAfstandSector[iSector] - 1),0.5);
}
for(iCylinder=0;iCylinder<=MAXCYLINDER;iCylinder++){
    stdevHoogteCylinder[iCylinder] = pow(stdevHoogteCylinder[iCylinder] / (aantalHoogteCylinder[iCylinder] - 1),0.5);
    stdevAfstandCylinder[iCylinder] = pow(stdevAfstandCylinder[iCylinder] / (aantalAfstandCylinder[iCylinder] - 1),0.5);
}

iAantalHoogte = 0;
p10Hoogte = -999.0;
p25Hoogte = -999.0;
p50Hoogte = -999.0;
p75Hoogte = -999.0;
p90Hoogte = -999.0;
iAantalAfstand = 0;
p10Afstand = -999.0;
p25Afstand = -999.0;
p50Afstand = -999.0;
p75Afstand = -999.0;
p90Afstand = -999.0;
for(iSector=0;iSector<=MAXSECTOR;iSector++){
    iAantalHoogteSector[iSector] = 0;
    p10HoogteSector[iSector] = -999.0;

```

```

p25HoogteSector[iSector] = -999.0;
p50HoogteSector[iSector] = -999.0;
p75HoogteSector[iSector] = -999.0;
p90HoogteSector[iSector] = -999.0;
iAantalAfstandSector[iSector] = 0;
p10AfstandSector[iSector] = -999.0;
p25AfstandSector[iSector] = -999.0;
p50AfstandSector[iSector] = -999.0;
p75AfstandSector[iSector] = -999.0;
p90AfstandSector[iSector] = -999.0;
for(iCylinder=0;iCylinder<=MAXCYLINDER;iCylinder++){
    iArea = iSector*10 + iCylinder;
    iAantalHoogteArea[iArea] = 0;
    p10HoogteArea[iArea] = -999.0;
    p25HoogteArea[iArea] = -999.0;
    p50HoogteArea[iArea] = -999.0;
    p75HoogteArea[iArea] = -999.0;
    p90HoogteArea[iArea] = -999.0;
    iAantalAfstandArea[iArea] = 0;
    p10AfstandArea[iArea] = -999.0;
    p25AfstandArea[iArea] = -999.0;
    p50AfstandArea[iArea] = -999.0;
    p75AfstandArea[iArea] = -999.0;
    p90AfstandArea[iArea] = -999.0;
    if(iSector==0) {
        iAantalHoogteCylinder[iCylinder] = 0;
        p10HoogteCylinder[iCylinder] = -999.0;
        p25HoogteCylinder[iCylinder] = -999.0;
        p50HoogteCylinder[iCylinder] = -999.0;
        p75HoogteCylinder[iCylinder] = -999.0;
        p90HoogteCylinder[iCylinder] = -999.0;
        iAantalAfstandCylinder[iCylinder] = 0;
        p10AfstandCylinder[iCylinder] = -999.0;
        p25AfstandCylinder[iCylinder] = -999.0;
        p50AfstandCylinder[iCylinder] = -999.0;
        p75AfstandCylinder[iCylinder] = -999.0;
        p90AfstandCylinder[iCylinder] = -999.0;
    }
}

```

```

    }
}
for(i=0;i<=MAXAREA;i++){
    iAantalHoogteArea[i] = 0;
    p10HoogteArea[i] = -999.0;
    p25HoogteArea[i] = -999.0;
    p50HoogteArea[i] = -999.0;
    p75HoogteArea[i] = -999.0;
    p90HoogteArea[i] = -999.0;
}
for(iSector=0;iSector<=MAXSECTOR;iSector++){
    iAantalHoogteSector[iSector] = 0;
    p10HoogteSector[iSector] = -999.0;
    p25HoogteSector[iSector] = -999.0;
    p50HoogteSector[iSector] = -999.0;
    p75HoogteSector[iSector] = -999.0;
    p90HoogteSector[iSector] = -999.0;
}
for(iCylinder=0;iCylinder<=MAXCYLINDER;iCylinder++){
    iAantalHoogteCylinder[iCylinder] = 0;
    p10HoogteCylinder[iCylinder] = -999.0;
    p25HoogteCylinder[iCylinder] = -999.0;
    p50HoogteCylinder[iCylinder] = -999.0;
    p75HoogteCylinder[iCylinder] = -999.0;
    p90HoogteCylinder[iCylinder] = -999.0;
}
for(k=0;k<=MAXHEIGHT;k++){
    for(i=0;i<=MAXAREA;i++){
        iSector = (i - (i % 10))/10;
        iCylinder = i % 10;
        if( i>10 && (i % 10 == 1 || i % 10 == 2 || i % 10 == 3) ) {
            for(j=0;j<=MAXCODES;j++){
                // Hoogte
                // Algemeen
                iAantalHoogte += count_hint[i][j][k];
                if(iAantalHoogte>0.1*aantalHoogteTotaal && p10Hoogte<-99.0) p10Hoogte = k/100.0;
                if(iAantalHoogte>0.25*aantalHoogteTotaal && p25Hoogte<-99.0) p25Hoogte = k/100.0;
                if(iAantalHoogte>0.5*aantalHoogteTotaal && p50Hoogte<-99.0) p50Hoogte = k/100.0;
            }
        }
    }
}

```

```

if(iAantalHoogte>0.75*aantalHoogteTotaal && p75Hoogte<-99.0) p75Hoogte = k/100.0;
if(iAantalHoogte>0.9*aantalHoogteTotaal && p90Hoogte<-99.0) p90Hoogte = k/100.0;
// Sector
iAantalHoogteSector[iSector] += count_hint[i][j][k];
if(iAantalHoogteSector[iSector]>0.1*aantalHoogteSector[iSector] && p10HoogteSector[iSector]<-99.0)
    p10HoogteSector[iSector] = k/100.0;
if(iAantalHoogteSector[iSector]>0.25*aantalHoogteSector[iSector] && p25HoogteSector[iSector]<-99.0)
    p25HoogteSector[iSector] = k/100.0;
if(iAantalHoogteSector[iSector]>0.5*aantalHoogteSector[iSector] && p50HoogteSector[iSector]<-99.0)
    p50HoogteSector[iSector] = k/100.0;
if(iAantalHoogteSector[iSector]>0.75*aantalHoogteSector[iSector] && p75HoogteSector[iSector]<-99.0)
    p75HoogteSector[iSector] = k/100.0;
if(iAantalHoogteSector[iSector]>0.9*aantalHoogteSector[iSector] && p90HoogteSector[iSector]<-99.0)
    p90HoogteSector[iSector] = k/100.0;
// Cylinder
iAantalHoogteCylinder[iCylinder] += count_hint[i][j][k];
if(iAantalHoogteCylinder[iCylinder]>0.1*aantalHoogteCylinder[iCylinder] && p10HoogteCylinder[iCylinder]<-99.0)
    p10HoogteCylinder[iCylinder] = k/100.0;
if(iAantalHoogteCylinder[iCylinder]>0.25*aantalHoogteCylinder[iCylinder] && p25HoogteCylinder[iCylinder]<-99.0)
    p25HoogteCylinder[iCylinder] = k/100.0;
if(iAantalHoogteCylinder[iCylinder]>0.5*aantalHoogteCylinder[iCylinder] && p50HoogteCylinder[iCylinder]<-99.0)
    p50HoogteCylinder[iCylinder] = k/100.0;
if(iAantalHoogteCylinder[iCylinder]>0.75*aantalHoogteCylinder[iCylinder] && p75HoogteCylinder[iCylinder]<-99.0)
    p75HoogteCylinder[iCylinder] = k/100.0;
if(iAantalHoogteCylinder[iCylinder]>0.9*aantalHoogteCylinder[iCylinder] && p90HoogteCylinder[iCylinder]<-99.0)
    p90HoogteCylinder[iCylinder] = k/100.0;
// Area
iAantalHoogteArea[i] += count_hint[i][j][k];
if(iAantalHoogteArea[i]>0.1*aantalHoogteArea[i] && p10HoogteArea[i]<-99.0) p10HoogteArea[i] = k/100.0;
if(iAantalHoogteArea[i]>0.25*aantalHoogteArea[i] && p25HoogteArea[i]<-99.0) p25HoogteArea[i] = k/100.0;
if(iAantalHoogteArea[i]>0.5*aantalHoogteArea[i] && p50HoogteArea[i]<-99.0) p50HoogteArea[i] = k/100.0;
if(iAantalHoogteArea[i]>0.75*aantalHoogteArea[i] && p75HoogteArea[i]<-99.0) p75HoogteArea[i] = k/100.0;
if(iAantalHoogteArea[i]>0.9*aantalHoogteArea[i] && p90HoogteArea[i]<-99.0) p90HoogteArea[i] = k/100.0;

// Afstand
// Algemeen
iAantalAfstand += aantal_afstand[i][j][k];
if(iAantalAfstand>0.1*aantalAfstandTotaal && p10Afstand<-99.0) p10Afstand = k*10.0;

```

```

if(iAantalAfstand>0.25*aantalAfstandTotaal && p25Afstand<-99.0) p25Afstand = k*10.0;
if(iAantalAfstand>0.5*aantalAfstandTotaal && p50Afstand<-99.0) p50Afstand = k*10.0;
if(iAantalAfstand>0.75*aantalAfstandTotaal && p75Afstand<-99.0) p75Afstand = k*10.0;
if(iAantalAfstand>0.9*aantalAfstandTotaal && p90Afstand<-99.0) p90Afstand = k*10.0;
// Sector
iAantalAfstandSector[iSector] += aantal_afstand[i][j][k];
if(iAantalAfstandSector[iSector]>0.1*aantalAfstandSector[iSector] && p10AfstandSector[iSector]<-99.0)
    p10AfstandSector[iSector] = k*10.0;
if(iAantalAfstandSector[iSector]>0.25*aantalAfstandSector[iSector] && p25AfstandSector[iSector]<-99.0)
    p25AfstandSector[iSector] = k*10.0;
if(iAantalAfstandSector[iSector]>0.5*aantalAfstandSector[iSector] && p50AfstandSector[iSector]<-99.0)
    p50AfstandSector[iSector] = k*10.0;
if(iAantalAfstandSector[iSector]>0.75*aantalAfstandSector[iSector] && p75AfstandSector[iSector]<-99.0)
    p75AfstandSector[iSector] = k*10.0;
if(iAantalAfstandSector[iSector]>0.9*aantalAfstandSector[iSector] && p90AfstandSector[iSector]<-99.0)
    p90AfstandSector[iSector] = k*10.0;
// Cylinder
iAantalAfstandCylinder[iCylinder] += aantal_afstand[i][j][k];
if(iAantalAfstandCylinder[iCylinder]>0.1*aantalAfstandCylinder[iCylinder] && p10AfstandCylinder[iCylinder]<-99.0)
    p10AfstandCylinder[iCylinder] = k*10.0;
if(iAantalAfstandCylinder[iCylinder]>0.25*aantalAfstandCylinder[iCylinder] && p25AfstandCylinder[iCylinder]<-99.0)
    p25AfstandCylinder[iCylinder] = k*10.0;
if(iAantalAfstandCylinder[iCylinder]>0.5*aantalAfstandCylinder[iCylinder] && p50AfstandCylinder[iCylinder]<-99.0)
    p50AfstandCylinder[iCylinder] = k*10.0;
if(iAantalAfstandCylinder[iCylinder]>0.75*aantalAfstandCylinder[iCylinder] && p75AfstandCylinder[iCylinder]<-99.0)
    p75AfstandCylinder[iCylinder] = k*10.0;
if(iAantalAfstandCylinder[iCylinder]>0.9*aantalAfstandCylinder[iCylinder] && p90AfstandCylinder[iCylinder]<-99.0)
    p90AfstandCylinder[iCylinder] = k*10.0;
// Area
iAantalAfstandArea[i] += aantal_afstand[i][j][k];
if(iAantalAfstandArea[i]>0.1*aantalAfstandArea[i] && p10AfstandArea[i]<-99.0) p10AfstandArea[i] = k*10.0;
if(iAantalAfstandArea[i]>0.25*aantalAfstandArea[i] && p25AfstandArea[i]<-99.0) p25AfstandArea[i] = k*10.0;
if(iAantalAfstandArea[i]>0.5*aantalAfstandArea[i] && p50AfstandArea[i]<-99.0) p50AfstandArea[i] = k*10.0;
if(iAantalAfstandArea[i]>0.75*aantalAfstandArea[i] && p75AfstandArea[i]<-99.0) p75AfstandArea[i] = k*10.0;
if(iAantalAfstandArea[i]>0.9*aantalAfstandArea[i] && p90AfstandArea[i]<-99.0) p90AfstandArea[i] = k*10.0;
}
}

```

```

    }
}

// print output
printf("Opslaan resultaten ...\n");
fp = fopen(resfilename,"w");

fprintf(fp, "Punt:\t%s\n", pointname);
fprintf(fp, "Totaal\n");
fprintf(fp, "Hoogte:\n");
fprintf(fp, "aantalHoogteTotaal:\t%ld\n", aantalHoogteTotaal);
fprintf(fp, "somHoogte:\t%.3f\n", somHoogte);
fprintf(fp, "gemHoogte:\t%.3f\n", gemHoogte);
fprintf(fp, "stdevHoogte:\t%.3f\n", stdevHoogte);
fprintf(fp, "minHoogte:\t%.3f\n", minHoogte);
fprintf(fp, "p10Hoogte:\t%.3f\n", p10Hoogte);
fprintf(fp, "p25Hoogte:\t%.3f\n", p25Hoogte);
fprintf(fp, "p50Hoogte:\t%.3f\n", p50Hoogte);
fprintf(fp, "p75Hoogte:\t%.3f\n", p75Hoogte);
fprintf(fp, "p90Hoogte:\t%.3f\n", p90Hoogte);
fprintf(fp, "maxHoogte:\t%.3f\n", maxHoogte);
fprintf(fp, "Afstand:\n");
fprintf(fp, "aantalAfstandTotaal:\t%ld\n", aantalAfstandTotaal);
fprintf(fp, "somAfstand:\t%.2f\n", somAfstand);
fprintf(fp, "gemAfstand:\t%.2f\n", gemAfstand);
fprintf(fp, "stdevAfstand:\t%.2f\n", stdevAfstand);
fprintf(fp, "minAfstand:\t%.2f\n", minAfstand);
fprintf(fp, "p10Afstand:\t%.2f\n", p10Afstand);
fprintf(fp, "p25Afstand:\t%.2f\n", p25Afstand);
fprintf(fp, "p50Afstand:\t%.2f\n", p50Afstand);
fprintf(fp, "p75Afstand:\t%.2f\n", p75Afstand);
fprintf(fp, "p90Afstand:\t%.2f\n", p90Afstand);
fprintf(fp, "maxAfstand:\t%.2f\n\n", maxAfstand);
fprintf(fp, "\n\n");

fprintf(fp, "Aantallen per Sector en Area\n");
fprintf(fp, "Hoogte\n");

```

```

fprintf(fp, "Sector\tTotaal\t1\t2\t3\n");
for(i=0;i<=MAXAREA;i++){
    iSector = (i - (i % 10))/10;
    if( i>10 && (i % 10 == 1 || i % 10 == 2 || i % 10 == 3) ) {
        if(i % 10 == 1) {
            fprintf(fp, "%d\t%ld", iSector, aantalHoogteSector[iSector] );
        }
        fprintf(fp, "\t%ld", count_area[i]);
        if(i % 10 == 3) {
            fprintf(fp, "\n");
        }
    }
}
fprintf(fp, "\n");
fprintf(fp, "Afstand\n");
fprintf(fp, "Sector\tTotaal\t1\t2\t3\n");
for(i=0;i<=MAXAREA;i++){
    iSector = (i - (i % 10))/10;
    if( i>10 && (i % 10 == 1 || i % 10 == 2 || i % 10 == 3) ) {
        if(i % 10 == 1) {
            fprintf(fp, "%d\t%ld", iSector, aantalAfstandSector[iSector] );
        }
        fprintf(fp, "\t%ld", aantalAfstandArea[i]);
        if(i % 10 == 3) {
            fprintf(fp, "\n");
        }
    }
}
fprintf(fp, "\n\n");
fprintf(fp, "Code\tAantal\t\n");
for(i=0;i<=MAXCODES;i++){
    fprintf(fp, "%d\t%ld\n", i, count_code[i]);
}
fprintf(fp, "\n\n");
fprintf(fp, "Aantal waarden per Cylinder\n");
fprintf(fp, "Hoogte\n");
fprintf(fp, "Cylinder\tHoogte_Aantal\tHoogte_Gemiddelde\tHoogte_Stdev\tHoogte_min\tHoogte_p10\tHoogte_p25\tHoogte_p50\tHoogte_p75\tHoogte_p90\tHoogte_max\n");

```

```

for(iCylinder=1;iCylinder<=MAXCYLINDER;iCylinder++){
    fprintf(fp,"%d\t%ld", iCylinder, aantalHoogteCylinder[iCylinder]);
    if(aantalHoogteCylinder[iCylinder] > 0) {
        fprintf(fp,"\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f", gemHoogteCylinder[iCylinder], stdevHoogteCylinder[iCylinder],
minHoogteCylinder[iCylinder], p10HoogteCylinder[iCylinder], p25HoogteCylinder[iCylinder], p50HoogteCylinder[iCylinder], p75HoogteCylinder[iCylinder],
p90HoogteCylinder[iCylinder], maxHoogteCylinder[iCylinder]);
    }
    fprintf(fp,"\n");
}
fprintf(fp,"\n");
fprintf(fp,"Afstand\n");
fprintf(fp,"Cylinder\tAfstand_Aantal\tAfstand_Gemiddelde\tAfstand_Stdev\tAfstand_min\tAfstand_p10\tAfstand_p25\tAfstand_p50\tAfstand_p75\tAfstand
_p90\tAfstand_max\n");
for(iCylinder=1;iCylinder<=MAXCYLINDER;iCylinder++){
    fprintf(fp,"%d\t%ld", iCylinder, aantalAfstandCylinder[iCylinder]);
    if(aantalAfstandCylinder[iCylinder] > 0) {
        fprintf(fp,"\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f", gemAfstandCylinder[iCylinder], stdevAfstandCylinder[iCylinder],
minAfstandCylinder[iCylinder], p10AfstandCylinder[iCylinder], p25AfstandCylinder[iCylinder], p50AfstandCylinder[iCylinder], p75AfstandCylinder[iCylinder],
p90AfstandCylinder[iCylinder], maxAfstandCylinder[iCylinder]);
    }
    fprintf(fp,"\n");
}
fprintf(fp,"\n\n");

fprintf(fp,"Aantal waarden per Sector, Area, Code\n");
fprintf(fp,"Hoogte\n");
fprintf(fp,"Sector\tArea\tCode\tHoogte_Aantal\tHoogte_Gemiddelde\tHoogte_Stdev\tHoogte_min\tHoogte_p10\tHoogte_p25\tHoogte_p50\tHoogte_p75\tH
oogte_p90\tHoogte_max\n");
for(i=0;i<=MAXAREA;i++){
    iSector = (i - (i % 10))/10;
    if( i>10 && (i % 10 == 1 || i % 10 == 2 || i % 10 == 3) ) {
        // Sector
        if(i % 10 == 1) {
            fprintf(fp,"%d\t\t\t%ld", iSector, aantalHoogteSector[iSector]);
            if(aantalHoogteSector[iSector] > 0) {
                fprintf(fp,"\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f", gemHoogteSector[iSector],
stdevHoogteSector[iSector], minHoogteSector[iSector], p10HoogteSector[iSector], p25HoogteSector[iSector], p50HoogteSector[iSector],
p75HoogteSector[iSector], p90HoogteSector[iSector], maxHoogteSector[iSector]);
            }
        }
    }
}

```

```

    }
    fprintf(fp, "\n");
}
// Sector, Area
fprintf(fp, "%d\t%d\t%d\t%d", iSector, i % 10, aantalHoogteArea[i]);
if(aantalHoogteArea[i] > 0) {
    fprintf(fp, "\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f", gemHoogteArea[i], stdevHoogteArea[i], minHoogteArea[i],
p10HoogteArea[i], p25HoogteArea[i], p50HoogteArea[i], p75HoogteArea[i], p90HoogteArea[i], maxHoogteArea[i]);
}
fprintf(fp, "\n");
// Sector, Area, Code
for(j=0;j<=MAXCODES;j++){

    for(k=0;k<=MAXHEIGHT;k++){
        if(count_hint[i][j][k]>0) {
            stdevHoogteAreaCode[i][j] += count_hint[i][j][k] * pow((gemHoogteAreaCode[i][j] - k/100.0),2.0);
        }
    }
    stdevHoogteAreaCode[i][j] = pow(stdevHoogteAreaCode[i][j] / (aantalHoogteAreaCode[i][j] - 1),0.5);
    fprintf(fp, "%d\t%d\t%d\t%d\t%d", iSector, i % 10, j, aantalHoogteAreaCode[i][j]);
    if(aantalHoogteAreaCode[i][j] > 1) {
        fprintf(fp, "\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f", gemHoogteAreaCode[i][j],
stdevHoogteAreaCode[i][j], minHoogteAreaCode[i][j], p10HoogteAreaCode[i][j], p25HoogteAreaCode[i][j], p50HoogteAreaCode[i][j], p75HoogteAreaCode[i][j],
p90HoogteAreaCode[i][j], maxHoogteAreaCode[i][j]);
    }
    else {
        if (aantalHoogteAreaCode[i][j] > 0) {
            fprintf(fp, "\t%.3f", gemHoogteAreaCode[i][j]);
        }
    }
    fprintf(fp, "\n");
}
}
}
fprintf(fp, "\n");
fprintf(fp, "Afstand\n");
fprintf(fp, "Sector\tArea\tCode\tAfstand_Aantal\tAfstand_Gemiddelde\tAfstand_Stdev\tAfstand_min\tAfstand_p10\tAfstand_p25\tAfstand_p50\tAfstand_p7
5\tAfstand_p90\tAfstand_max\n");

```



```

        }
    }
    fprintf(fp, "\n");
}

time(&EndTime);
fprintf(fp, "Einde programma: verstreken tijd=%ld\n", EndTime-BeginTime);
fclose(fp);

printf("Einde programma: verstreken tijd=%ld\n", EndTime-BeginTime);
}

double **LeesAsciiGridDbl(char *FileName) {
    FILE *fp;
    char tempstr[MaxLineLength];
    int nCols, nRows, CellSize;
    int MissingValue;
    long xllcorner, yllcorner;
    long AantalWoorden;
    char *Woorden[MaxWordsPerLine];
    long i, j;
    double val;
    int valint;
    double **arrpp;

    if(strlen(FileName) == 0) {
        printf("%s", "Inlezen Ascii-grid: File om uit te lezen: ");
        gets(FileName);
        if(strlen(FileName) == 0) {
            exit(0);
        } else {
            printf("Gegevens worden gelezen uit file '%s\n'", FileName );
        }
    }
}

```

```

nCols=0;nRows=0;xllcorner=0;yllcorner=0;CellSize=0;
fp = fopen(FileName,"r");
fgets(tempstr,MaxLineLength,fp);
tempstr[strlen(tempstr)-1]= '\0';
AantalWoorden = splitstring(Woorden,tempstr);
nCols = atol(Woorden[1]);
fgets(tempstr,MaxLineLength,fp); tempstr[strlen(tempstr)-1]= '\0'; AantalWoorden = splitstring(Woorden,tempstr); nRows = atol(Woorden[1]);
fgets(tempstr,MaxLineLength,fp); tempstr[strlen(tempstr)-1]= '\0'; AantalWoorden = splitstring(Woorden,tempstr); xllcorner = atol(Woorden[1]);
fgets(tempstr,MaxLineLength,fp); tempstr[strlen(tempstr)-1]= '\0'; AantalWoorden = splitstring(Woorden,tempstr); yllcorner = atol(Woorden[1]);
fgets(tempstr,MaxLineLength,fp); tempstr[strlen(tempstr)-1]= '\0'; AantalWoorden = splitstring(Woorden,tempstr); CellSize = atol(Woorden[1]);
fgets(tempstr,MaxLineLength,fp); tempstr[strlen(tempstr)-1]= '\0'; AantalWoorden = splitstring(Woorden,tempstr); MissingValue = atol(Woorden[1]);

// allocate memory
arrpp = doublematrix(0,nRows,0,nCols); // new int[nRows*nCols];

for(i = 0; i<nRows; i++) {
    fgets(tempstr,MaxLineLength,fp);
    AantalWoorden = splitstring(Woorden,tempstr);
    for(j=0; j<nCols; j++) {
        val = atof(Woorden[j]);
        valint = atoi(Woorden[j]);
        if(valint != MissingValue) {
            arrpp[i][j] = atof(Woorden[j]);
        }
        else {
            arrpp[i][j] = (double)MISSINGVALUE;
        }
    }
}
fclose(fp);
return arrpp;
}

char **LeesAsciiGridByte(char *FileName) {
    FILE *fp;
    char tempstr[MaxLineLength];

```

```

int nCols, nRows, CellSize;
int MissingValue;
long xllcorner, yllcorner;
long AantalWoorden;
char *Woorden[MaxWordsPerLine];
long i, j;
char val;
short valshort;
char **arrpp;

if(strlen(FileName) == 0) {
    printf("%s", "Inlezen Ascii-grid: File om uit te lezen: ");
    gets(FileName);
    if(strlen(FileName) == 0) {
        exit(0);
    } else {
        printf("Gegevens worden gelezen uit file '%s\n'", FileName );
    }
}

nCols=0;nRows=0;xllcorner=0;yllcorner=0;CellSize=0;
fp = fopen(FileName, "r");
fgets(tempstr,MaxLineLength,fp);
tempstr[strlen(tempstr)-1]= '\0';
AantalWoorden = splitstring(Woorden,tempstr);
nCols = atol(Woorden[1]);
fgets(tempstr,MaxLineLength,fp); tempstr[strlen(tempstr)-1]= '\0'; AantalWoorden = splitstring(Woorden,tempstr); nRows = atol(Woorden[1]);
fgets(tempstr,MaxLineLength,fp); tempstr[strlen(tempstr)-1]= '\0'; AantalWoorden = splitstring(Woorden,tempstr); xllcorner = atol(Woorden[1]);
fgets(tempstr,MaxLineLength,fp); tempstr[strlen(tempstr)-1]= '\0'; AantalWoorden = splitstring(Woorden,tempstr); yllcorner = atol(Woorden[1]);
fgets(tempstr,MaxLineLength,fp); tempstr[strlen(tempstr)-1]= '\0'; AantalWoorden = splitstring(Woorden,tempstr); CellSize = atol(Woorden[1]);
fgets(tempstr,MaxLineLength,fp); tempstr[strlen(tempstr)-1]= '\0'; AantalWoorden = splitstring(Woorden,tempstr); MissingValue = atol(Woorden[1]);

// allocate memory
arrpp = charmatrix(0,nRows,0,nCols); // new int[nRows*nCols];

for(i = 0; i<nRows; i++) {
    fgets(tempstr,MaxLineLength,fp);
    AantalWoorden = splitstring(Woorden,tempstr);

```

```

        for(j=0; j<nCols; j++) {
            val = atoi(Woorden[j]);
            valshort = atoi(Woorden[j]);
            if(valshort != MissingValue) {
                arrpp[i][j] = (char)valshort;
            }
            else {
                arrpp[i][j] = MISSINGVALUE;
            }
        }
    }

fclose(fp);
return arrpp;
}

bool **LeesAsciiGridBool(char *FileName) {
    FILE *fp;
    char tempstr[MaxLineLength];
    int nCols, nRows, CellSize;
    int MissingValue;
    long xllcorner, yllcorner;
    long AantalWoorden;
    char *Woorden[MaxWordsPerLine];
    long i, j, val;
    bool **arrpp;

    if(strlen(FileName) == 0) {
        printf("%s", "Inlezen Ascii-grid: File om uit te lezen: ");
        gets(FileName);
        if(strlen(FileName) == 0) {
            exit(0);
        } else {
            printf("Gegevens worden gelezen uit file '%s\n'", FileName );
        }
    }
}

```

```

nCols=0;nRows=0;xllcorner=0;yllcorner=0;CellSize=0;
fp = fopen(FileName,"r");
fgets(tempstr,MaxLineLength,fp);
tempstr[strlen(tempstr)-1]= '\0';
AantalWoorden = splitstring(Woorden,tempstr);
nCols = atoi(Woorden[1]);
fgets(tempstr,MaxLineLength,fp); tempstr[strlen(tempstr)-1]= '\0'; AantalWoorden = splitstring(Woorden,tempstr); nRows = atoi(Woorden[1]);
fgets(tempstr,MaxLineLength,fp); tempstr[strlen(tempstr)-1]= '\0'; AantalWoorden = splitstring(Woorden,tempstr); xllcorner = atoi(Woorden[1]);
fgets(tempstr,MaxLineLength,fp); tempstr[strlen(tempstr)-1]= '\0'; AantalWoorden = splitstring(Woorden,tempstr); yllcorner = atoi(Woorden[1]);
fgets(tempstr,MaxLineLength,fp); tempstr[strlen(tempstr)-1]= '\0'; AantalWoorden = splitstring(Woorden,tempstr); CellSize = atoi(Woorden[1]);
fgets(tempstr,MaxLineLength,fp); tempstr[strlen(tempstr)-1]= '\0'; AantalWoorden = splitstring(Woorden,tempstr); MissingValue = atoi(Woorden[1]);

// allocate memory
arrpp = boolmatrix(0,nRows,0,nCols); // new int[nRows*nCols];

for(i = 0; i<nRows; i++) {
    fgets(tempstr,MaxLineLength,fp);
    AantalWoorden = splitstring(Woorden,tempstr);
    for(j=0; j<nCols; j++) {
        val = atoi(Woorden[j]);
        if(val != MissingValue) {
            if(val==1) {
                arrpp[i][j] = true;
            }
            else {
                arrpp[i][j] = false;
            }
        }
        else {
            arrpp[i][j] = false;
        }
    }
}
fclose(fp);
return arrpp;
}

```

```

int **LeesAsciiGridInt(char *FileName, double ConversionFactor) {
    FILE *fp;
    char tempstr[MaxLineLength];
    int nCols, nRows, CellSize;
    int MissingValue;
    long xllcorner, yllcorner;
    long AantalWoorden;
    char *Woorden[MaxWordsPerLine];
    long i, j;
    int valint;
    int **arrpp;

    if(strlen(FileName) == 0) {
        printf("%s", "Inlezen Ascii-grid: File om uit te lezen: ");
        gets(FileName);
        if(strlen(FileName) == 0) {
            exit(0);
        } else {
            printf("Gegevens worden gelezen uit file '%s\n'", FileName );
        }
    }

    nCols=0;nRows=0;xllcorner=0;yllcorner=0;CellSize=0;
    fp = fopen(FileName, "r");
    fgets(tempstr,MaxLineLength,fp);
    tempstr[strlen(tempstr)-1]= '\0';
    AantalWoorden = splitstring(Woorden,tempstr);
    nCols = atol(Woorden[1]);
    fgets(tempstr,MaxLineLength,fp); tempstr[strlen(tempstr)-1]= '\0'; AantalWoorden = splitstring(Woorden,tempstr); nRows = atol(Woorden[1]);
    fgets(tempstr,MaxLineLength,fp); tempstr[strlen(tempstr)-1]= '\0'; AantalWoorden = splitstring(Woorden,tempstr); xllcorner = atol(Woorden[1]);
    fgets(tempstr,MaxLineLength,fp); tempstr[strlen(tempstr)-1]= '\0'; AantalWoorden = splitstring(Woorden,tempstr); yllcorner = atol(Woorden[1]);
    fgets(tempstr,MaxLineLength,fp); tempstr[strlen(tempstr)-1]= '\0'; AantalWoorden = splitstring(Woorden,tempstr); CellSize = atol(Woorden[1]);
    fgets(tempstr,MaxLineLength,fp); tempstr[strlen(tempstr)-1]= '\0'; AantalWoorden = splitstring(Woorden,tempstr); MissingValue = atol(Woorden[1]);

    // allocate memory
    arrpp = intmatrix(0,nRows,0,nCols);

    for(i = 0; i<nRows; i++) {

```

```

fgets(tempstr,MaxLineLength,fp);
    AantalWoorden = splitstring(Woorden,tempstr);
    for(j=0; j<nCols; j++) {
        valint = atoi(Woorden[j]);
        if(valint != MissingValue) {
            arrpp[i][j] = (int)((double)ConversionFactor*atof(Woorden[j]));
        }
        else {
            arrpp[i][j] = (int)MISSINGVALUE;
        }
    }
}
fclose(fp);
return arrpp;
}

```

```

short **LeesAsciiGridShort(char *FileName, double ConversionFactor) {
    FILE *fp;
    char tempstr[MaxLineLength];
    int nCols, nRows, CellSize;
    int MissingValue;
    long xllcorner, yllcorner;
    long AantalWoorden;
    char *Woorden[MaxWordsPerLine];
    long i, j;
    int valint;
    short **arrpp;

    if(strlen(FileName) == 0) {
        printf("%s", "Inlezen Ascii-grid: File om uit te lezen: ");
        gets(FileName);
        if(strlen(FileName) == 0) {
            exit(0);
        } else {
            printf("Gegevens worden gelezen uit file '%s\n'", FileName );
        }
    }
}

```

```

nCols=0;nRows=0;xllcorner=0;yllcorner=0;CellSize=0;
fp = fopen(FileName,"r");
fgets(tempstr,MaxLineLength,fp);
tempstr[strlen(tempstr)-1]= '\0';
AantalWoorden = splitstring(Woorden,tempstr);
nCols = atol(Woorden[1]);
fgets(tempstr,MaxLineLength,fp); tempstr[strlen(tempstr)-1]= '\0'; AantalWoorden = splitstring(Woorden,tempstr); nRows = atol(Woorden[1]);
fgets(tempstr,MaxLineLength,fp); tempstr[strlen(tempstr)-1]= '\0'; AantalWoorden = splitstring(Woorden,tempstr); xllcorner = atol(Woorden[1]);
fgets(tempstr,MaxLineLength,fp); tempstr[strlen(tempstr)-1]= '\0'; AantalWoorden = splitstring(Woorden,tempstr); yllcorner = atol(Woorden[1]);
fgets(tempstr,MaxLineLength,fp); tempstr[strlen(tempstr)-1]= '\0'; AantalWoorden = splitstring(Woorden,tempstr); CellSize = atol(Woorden[1]);
fgets(tempstr,MaxLineLength,fp); tempstr[strlen(tempstr)-1]= '\0'; AantalWoorden = splitstring(Woorden,tempstr); MissingValue = atol(Woorden[1]);

// allocate memory
arrpp = shortmatrix(0,nRows,0,nCols);

for(i = 0; i<nRows; i++) {
fgets(tempstr,MaxLineLength,fp);
    AantalWoorden = splitstring(Woorden,tempstr);
    for(j=0; j<nCols; j++) {
        valint = atoi(Woorden[j]);
        if(valint != MissingValue) {
            arrpp[i][j] = (short)(ConversionFactor*atof(Woorden[j]));
        }
        else {
            arrpp[i][j] = (short)MISSINGVALUE;
        }
    }
}
fclose(fp);
return arrpp;
}

int IsSpace(char chr) {
    if(chr == ' ') {
        return true;
    }
}

```

```

    }
    else {
        return false;
    }
}

```

```

long splitstring(char** arrstr, char* strInput) {
    char *p1, *p2;
    int length;
    int iWord;

    iWord = 0;
    length = strlen(strInput);

    for(p1=strInput; *p1 && IsSpace(*p1);p1++) {
    }

    if(*p1=='\0') {
        return(iWord);
    }
    while( *p1 ) {
        for(; *p1 && IsSpace(*p1);p1++) {
        }
        for(p2=p1+1; *p2 && !IsSpace(*p2);p2++) {
        }
        // p2 points now to first non-space character of next word or end of string
        if(iWord>MaxWordsPerLine) {
            *p1 = 0;
        }
        else {
            arrstr[iWord] = p1;
            if(*p2=='\0') {
                p1=p2;
            }
            else {
                p1=p2+1;
                *p2 = '\0';
            }
        }
    }
}

```

```

        }
        iWord++;
    }
}
return(iWord);
}

void nrerror(char error_text[]) {
    void exit(int x);

    fprintf(stderr,"Numerical runtime error...\n");
    fprintf(stderr,"%s\n", error_text);
    fprintf(stderr,"Now exiting the program...\n");
    getchar();
    exit(1);
}

double *doublevector(int nrl,int nrh) {
    double *m;

    m = (double*)malloc((unsigned) (nrh-nrl+1)*sizeof(double));
    if (!m) nrerror("allocation failure 1 in doublevector()");
    m -= nrl;

    return m;
}

double **doublematrix(int nrl,int nrh,int ncl,int nch) {
    int i;
    double **m;

    m = (double**)malloc((unsigned) (nrh-nrl+1)*sizeof(double*));
    if (!m) nrerror("allocation failure 1 in doublematrix()");
    m -= nrl;

```

```

for(i=nrl;i<nrh;i++) {
    m[i] = (double*)malloc( (unsigned) (nch-ncl+1)*sizeof(double));
    if (!m[i]) perror("allocation failure 2 in doublematrix()");
    m[i] -= ncl;
}
return m;
}

```

```

void free_doublematrix(double **m, int nrl, int nrh, int ncl, int nch) {
    int i;

    for (i=nrh; i>=nrl; i--) free( (char*) (m[i]+ncl) );
    free( (char*) (m+nrl) );

}

```

```

int **intmatrix(int nrl,int nrh,int ncl,int nch) {
    int i;
    int **m;

    m = (int**)malloc((unsigned) (nrh-nrl+1)*sizeof(int*));
    if (!m) perror("allocation failure 1 in intmatrix()");
    m -= nrl;

    for(i=nrl;i<nrh;i++) {
        m[i] = (int*)malloc( (unsigned) (nch-ncl+1)*sizeof(int));
        if (!m[i]) perror("allocation failure 2 in intmatrix()");
        m[i] -= ncl;
    }
    return m;
}

```

```

void free_intmatrix(int **m, int nrl, int nrh, int ncl, int nch) {
    int i;

    for (i=nrh; i>=nrl; i--) free( (char*) (m[i]+ncl) );
    free( (char*) (m+nrl) );
}

```

```

}

short **shortmatrix(int nrl,int nrh,int ncl,int nch) {
    int i;
    short **m;

    m = (short**)malloc((unsigned) (nrh-nrl+1)*sizeof(short));
    if (!m) nrerror("allocation failure 1 in shortmatrix()");
    m -= nrl;

    for(i=nrl;i<nrh;i++) {
        m[i] = (short*)malloc( (unsigned) (nch-ncl+1)*sizeof(short));
        if (!m[i]) nrerror("allocation failure 2 in shortmatrix()");
        m[i] -= ncl;
    }
    return m;
}

void free_shortmatrix(short **m, int nrl, int nrh, int ncl, int nch) {
    int i;

    for (i=nrh; i>=nrl; i--) free( (char*) (m[i]+ncl) );
    free( (char*) (m+nrl) );
}

bool **boolmatrix(int nrl,int nrh,int ncl,int nch) {
    int i;
    bool **m;

    m = (bool**)malloc((unsigned) (nrh-nrl+1)*sizeof(bool));
    if (!m) nrerror("allocation failure 1 in intmatrix()");
    m -= nrl;

    for(i=nrl;i<nrh;i++) {
        m[i] = (bool*)malloc( (unsigned) (nch-ncl+1)*sizeof(bool));
        if (!m[i]) nrerror("allocation failure 2 in intmatrix()");
        m[i] -= ncl;
    }
}

```

```

    }
    return m;
}

void free_boolmatrix(bool **m, int nrl, int nrh, int ncl, int nch) {
    int i;

    for (i=nrh; i>=nrl; i--) free( (char*) (m[i]+ncl) );
    free( (char*) (m+nrl) );
}

char **charmatrix(int nrl,int nrh,int ncl,int nch) {
    int i;
    char **m;

    m = (char**)malloc((unsigned) (nrh-nrl+1)*sizeof(char*));
    if (!m) perror("allocation failure 1 in longmatrix()");
    m -= nrl;

    for(i=nrl;i<nrh;i++) {
        m[i] = (char*)malloc( (unsigned) (nch-ncl+1)*sizeof(char));
        if (!m[i]) perror("allocation failure 2 in longmatrix()");
        m[i] -= ncl;
    }
    return m;
}

void free_charmatrix(char **m, int nrl, int nrh, int ncl, int nch) {
    int i;

    for (i=nrh; i>=nrl; i--) free( (char*) (m[i]+ncl) );
    free( (char*) (m+nrl) );
}

long *longvector(int nrl,int nrh) {
    long *m;

```

```

    m = (long*)malloc((unsigned) (nrh-nrl+1)*sizeof(long));
    if (!m) nrerror("allocation failure 1 in longvector()");
    m -= nrl;

    return m;
}

long **longmatrix(int nrl,int nrh,int ncl,int nch) {
    int i;
    long **m;

    m = (long**)malloc((unsigned) (nrh-nrl+1)*sizeof(long*));
    if (!m) nrerror("allocation failure 1 in longmatrix()");
    m -= nrl;

    for(i=nrl;i<nrh;i++) {
        m[i] = (long*)malloc( (unsigned) (nch-ncl+1)*sizeof(long));
        if (!m[i]) nrerror("allocation failure 2 in longmatrix()");
        m[i] -= ncl;
    }
    return m;
}

void free_longmatrix(long **m, int nrl, int nrh, int ncl, int nch) {
    int i;

    for (i=nrh; i>=nrl; i--) free( (char*) (m[i]+ncl) );
    free( (char*) (m+nrl) );
}

long ***long3Dmatrix(int nrl, int nrh, int ncl, int nch, int nsl, int nsh) {
    int i, j;
    long ***m;

    m = (long***)malloc((unsigned) (nrh-nrl+1)*sizeof(long**));
    if (!m) nrerror("allocation failure 1 in long3Dmatrix()");

```

```

m -= nrl;

for(i=nrl;i<nrh;i++) {
    m[i] = (long**)malloc( (unsigned) (nch-ncl+1)*sizeof(long*));
    if (!m[i]) nrerror("allocation failure 2 in long3Dmatrix()");
    m[i] -= ncl;

    for(j=ncl;j<nch;j++) {
        m[i][j] = (long*)malloc( (unsigned) (nsh-nsl+1)*sizeof(long));
        if (!m[i][j]) nrerror("allocation failure 3 in long3Dmatrix()");
        m[i][j] -= nsl;
    }

}
return m;
}

void free_long3Dmatrix(long ***m, int nrl, int nrh, int ncl, int nch, int nsl, int nsh) {
    int i, j;

    for (i=nrh-1; i>=nrl; i--) {
        for(j=nch-1; j>=ncl; j--) {
            free( (char*) (m[i][j]+nsl) );
        }
        free( (char*) (m[i]+ncl) );
    }

    free( (char*) (m+nrl) );
}

```