```c
/*
  read_grads2nc.c

  This is the main module of the "grads2nc" program.  It converts a GrADS
  control file (with associated IEEE data file) into a netCDF file.

  Written by Hans Vahlenkamp
  Geophysical Fluid Dynamics Laboratory/NOAA
  Princeton University Forrestal Campus
  Last updated: 10/28/98
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>
#include <sys/types.h>
#include <sys/stat.h>
#ifdef HOST_T90
#include <fortran.h>
#endif

#define MAXLINELEN 10000
#define MAXSTRLEN 256
#define MSB 1
#define LSB 2

#ifdef HOST_T90
extern WRITE_GRADS2NC(_fcd, _fcd, _fcd, float *, int *, int *, int *, int *,
                      float *, float *, float *, float *, int *, int *, int *,
                      int *, int *, _fcd, _fcd, float *, float *, int *, int *,
                      int *, _fcd, int *, int *, int *, int *, float *,
                      int *, _fcd, _fcd, _fcd, _fcd, int *, _fcd, _fcd, _fcd,
                      int *, int *, int *, int *, float *, float *, _f_log,
                      int *, int *);
#else
extern write_grads2nc_(char *, char *, char *, float *, int *, int *, int *,
                       int *, float *, float *, float *, float *, int *, int *,
                       int *, int *, int *, char *, char *, float *, float *,
                       int *, int *, int *, char *, int *, int *, int *, int *,
                       int *, float *, int *, char *, char *, char *, char *,
                       int *, char *, char *, char *, int *, int *, int *,
                       int *, float *, float *, int *, int *, int *);
#endif
int gaussian_lats(int, int, float *, int);


void main(int argc, char *argv[])
  {
   unsigned char netcdfffiospec=0;  /* Override the built-in value for the   */
                                    /* NETCDF_FFIOSPEC environment variable? */
   int calcrange=0;  /* Calculate the min and max values for the variables? */
   int verbose=0;  /* Verbose mode; print extensive progress information? */
   int quiet=0;  /* Quiet mode; no progress output */
   unsigned char merror=0;  /* Is there a parsing error? */
   FILE *ctlfile;  /* Pointer to the control file */
   unsigned char endian=0;  /* Endianness of the current machine */
   char binfile[MAXSTRLEN];                   /* IEEE data file */
   char newbinfile[MAXSTRLEN], *binfileptr;   /*        "       */
   struct stat binfileinfo;  /* Information about the data file */
   char gtitle[MAXSTRLEN];  /* Data set description */
   int x, y, z, t, v, c, d=0;  /* Loop variables */
   char ctlline[MAXLINELEN], *ctllineptr;  /* Current control file line */
   unsigned char ctlfdone=0;  /* Has ENDVARS (end of file) been reached? */
   char dtype[MAXSTRLEN];  /* Data organization: gridded, station, or GRIB */
   unsigned char misvalfound=0;  /* Was a missing value found? */
   float misval=0;  /* Data set's missing value */
   int yrev=0, zrev=0;  /* Flip the data along the Y and Z axes? */
   unsigned char bigendian=0, littleendian=0;  /* Endianness of the machine */
                                               /* that wrote the IEEE data  */
   int byteswapped=0;  /* Is the data byte-order different than what the */
                       /* machine that wrote it normally expects?        */
   int xlen=0, ylen=0, zlen=0, tlen=0;  /* Lengths of the 4 axes */
   float xstart=0, xincr=0, ystart=0, yincr=0;    /* Information for    */
   int gausystart;                                /* constructing       */
   float zstart=0, zincr=0;                       /* coordinate values */
   char tstart[MAXSTRLEN], tincr[MAXSTRLEN];      /*         "          */
   char tincr2[MAXSTRLEN], tstart2[MAXSTRLEN];    /*         "          */
   char tstart3[MAXSTRLEN], month[MAXSTRLEN];     /*         "          */
   int tincr1, hour, minute, day, year;           /*         "          */
   char months[12][5]={"JAN","FEB","MAR","APR",   /*         "          */
                       "MAY","JUN","JUL","AUG",   /*         "          */
                       "SEP","OCT","NOV","DEC"};  /*         "          */
   float *xvals=NULL, xval, *yvals=NULL, yval;    /*         "          */
   float *zvals=NULL, zval, *tvals=NULL;          /*         "          */
   int nvars=0;  /* Number of data variables */
   char *varnames=NULL, *varnamesptr;  /* Names of the data variables */
   char *vardescs=NULL, *vardescsptr;  /* Descriptions of the data variables */
   int *varzlens=NULL;  /* Length of each variable's Z dimension */
   int *varzdims=NULL;  /* Number of each variable's Z dimension */
   int *dimzlens=NULL;  /* Length of each final Z dimension */
   char varrec[MAXSTRLEN], *varrecptr;  /* One line of variable information */
   int varzlen=0;  /* Number of Z levels for a variable */
   char gribunits[MAXSTRLEN];  /* Special handling (GRIB) specification for */
                               /* a variable                                 */
   int nzdims=0, zmatch;  /* Number of different Z dimensions */
   char ncfile[128];  /* Output netCDF file */
   float *valuesxy=NULL;  /* Buffer equal to the size of one Z level */
   float *varranges=NULL;  /* Minimum and maximum variable data values */
   int mxdimout;  /* Maximum length of a dimension */
   int ndims;  /* Total number of dimensions */

   /* Arrays used by the Fortran function "write_grads2nc" for "ncsetup" */
   /* which are ordinarily declared in "nc_stufw.h"                       */
   float *dimvals;  /* Coordinate values along the dimensions */
   int *lendims;  /* Lengths of all the dimensions */
   char *cdimnam, *cdimlnam, *cdimunits;  /* Dimension names and units */
   char *cart_axis;  /* Cartesian axes that the dimensions represent */
   int *ipositive;  /* Indicates if coordinate values increase in the */
                    /* positive direction along the cartesian axes    */
   char *cvarnam, *cvarlnam, *cvarunits;  /* Variable names and units */
   int *nbits;  /* Number of bits per variable (always 32 in this case) */
   int *idimindx;  /* Dimension indices for each variable */
   int *nvdims;  /* Number of dimensions for each variable */
   int *maxnitems;  /* Information for time-averaged data (not used) */
   float *validr;  /* Valid numeric range for each variable */
   float *vspval;  /* Special value for each variable */
   int *lspval;  /* Does each variable have a special value attribute? */
   int *istart, *icount;  /* Range of dimension indices for "ncstore" */

   /* Check the arguments */
   if (argc < 3)
     {
      fprintf(stderr,"Usage: grads2nc infile.ctl outfile.nc [-n] [-r] [-v | -q]\n\n");
      fprintf(stderr,"  -n   Use the shell's setting for \"NETCDF_FFIOSPEC\", otherwis
e it will use\n");
      fprintf(stderr,"       the built-in default of \"cachea:896:2\" (Cray only).\n")
;
      fprintf(stderr,"  -r   Finds the actual minimum and maximum values for each vari
```

```c
able instead\n");
      fprintf(stderr,"       of using the default \"valid_range\" of -1e+37 to 1e+37.\n");
      fprintf(stderr,"  -v   Verbose mode; print extensive progress information.\n");
      fprintf(stderr,"  -q   Quiet mode; no progress output.\n\n");
      fprintf(stderr,"*This program assumes the control file format for GrADS 1.5.x; older or newer\n");
      fprintf(stderr," versions may or may not parse correctly, but they will probably be fine.\n");
      fprintf(stderr," Multi-file templates, station data, GRIB data, and initial file headers are\n");
      fprintf(stderr," currently not supported.  The netCDF X and Y dimensions are given a \"units\"\n");
      fprintf(stderr," attribute of \"degrees_E\" and \"degrees_N\" respectively.  All the Z dimensions\n");
      fprintf(stderr," are given units of \"level\".  All the variables are given units of \"unknown\"\n");
      fprintf(stderr," since there is no consistent way of extracting this information from the\n");
      fprintf(stderr," GrADS control file.\n");
      exit(1);
     }
   if (argc > 3)
     {
      if (!strcmp(argv[3],"-n")) netcdfffiospec=1;
      else if (!strcmp(argv[3],"-r")) calcrange=1;
      else if (!strcmp(argv[3],"-v")) verbose=1;
      else if (!strcmp(argv[3],"-q")) quiet=1;
     }
   if (argc > 4)
     {
      if (!strcmp(argv[4],"-n")) netcdfffiospec=1;
      else if (!strcmp(argv[4],"-r")) calcrange=1;
      else if (!strcmp(argv[4],"-v")) verbose=1;
      else if (!strcmp(argv[4],"-q")) quiet=1;
     }
   if (argc > 5)
     {
      if (!strcmp(argv[5],"-n")) netcdfffiospec=1;
      else if (!strcmp(argv[5],"-r")) calcrange=1;
      else if (!strcmp(argv[5],"-v")) verbose=1;
      else if (!strcmp(argv[5],"-q")) quiet=1;
     }
   if (argc > 6)
     {
      if (!strcmp(argv[6],"-n")) netcdfffiospec=1;
      else if (!strcmp(argv[6],"-r")) calcrange=1;
      else if (!strcmp(argv[6],"-v")) verbose=1;
      else if (!strcmp(argv[6],"-q")) quiet=1;
     }

   /* If grads2nc is running on a Cray system then set the "NETCDF_FFIOSPEC" */
   /* environment variable for greater read/write efficiency               */
#ifdef HOST_T90
   if (netcdfffiospec && getenv("NETCDF_FFIOSPEC")!=NULL)
     {
      if (!quiet)
        printf("Using the current value of the \"NETCDF_FFIOSPEC\" environment variable.\n\n");
     }
   else
     {
      if (putenv("NETCDF_FFIOSPEC=cachea:896:2"))
        fprintf(stderr,"Error setting the \"NETCDF_FFIOSPEC\" environment variable!\n");
```

```c
      else if (!quiet)
        printf("Setting the \"NETCDF_FFIOSPEC\" environment variable to \"cachea:896:2\".\n\n");
     }
#endif

   /* Open the control file */
   if ((ctlfile=fopen(argv[1],"ra"))==NULL)
     {
      fprintf(stderr,"Cannot open the GrADS control file:\n  %s\n",argv[1]);
      exit(1);
     }

   /* Get the endianness (MSB or LSB) of the current machine */
#ifdef BIG_ENDIAN
   endian+=MSB;
#endif
#ifdef LITTLE_ENDIAN
   endian+=LSB;
#endif
   if (!endian || endian > 2)
     {
      fprintf(stderr,"Either \"BIG_ENDIAN\" or \"LITTLE_ENDIAN\" must be defined in the makefile!\n");
      exit(1);
     }

   /* Parse the record types */
   if (!quiet)
     {
      printf("Parsing GrADS control file...\n");
      printf("-----------------------------\n\n");
     }
   binfile[0]='\0'; gtitle[0]='\0';
   while (fgets(ctlline,MAXLINELEN,ctlfile)!=NULL)
     {
      if (ctlline[strlen(ctlline)-1]!='\n')
        {
         fprintf(stderr,"Control file line is too long; maximum is %d characters!\n",
                 MAXLINELEN);
         merror=1; break;
        }
      if (ctlfdone)
        {
         if ((ctllineptr=strtok(ctlline," \n"))!=NULL)
           fprintf(stderr,"\nInformation after \"ENDVARS\" will be ignored!\n");
         break;
        }
      for (c=0; c < strlen(ctlline); c++)
        if ((isspace(ctlline[c]) || iscntrl(ctlline[c])) && ctlline[c]!='\n')
          ctlline[c]=' ';
      if ((ctllineptr=strtok(ctlline," \n"))==NULL) continue;
      if (!strcasecmp(ctllineptr,"DSET"))
        {
         if ((ctllineptr=strtok(NULL," \n"))==NULL)
           {
            merror=1; break;
           }
         strcpy(binfile,ctllineptr);
         if (binfile[0]=='^')
           {
            binfile[0]='/'; strcpy(newbinfile,argv[1]);
            if ((binfileptr=strrchr(newbinfile,'/'))!=NULL)
              {
               *binfileptr='\0'; strcat(newbinfile,binfile);
```

```
            }
           else
             sprintf(newbinfile,".%s",binfile);
           strcpy(binfile,newbinfile);
           }
        if (!quiet) printf("binfile = %s\n",binfile);
        }
      else if (!strcasecmp(ctllineptr,"DTYPE"))
        {
        if ((ctllineptr=strtok(NULL," \n"))==NULL) continue;
        strcpy(dtype,ctllineptr);
        if (!strcasecmp(dtype,"STATION"))
          {
           fprintf(stderr,"Station data is not supported!\n"); merror=1;
           break;
          }
        else if (!strcasecmp(dtype,"GRIB"))
          {
           fprintf(stderr,"GRIB data is not supported!\n"); merror=1;
           break;
          }
        }
      else if (!strcasecmp(ctllineptr,"STNMAP"))
        {
        if ((ctllineptr=strtok(NULL," \n"))==NULL) continue;
        fprintf(stderr,"Station map file is not supported!\n"); merror=1;
        break;
        }
      else if (!strcasecmp(ctllineptr,"FILEHEADER"))
        {
        if ((ctllineptr=strtok(NULL," \n"))==NULL) continue;
        fprintf(stderr,"Initial file header is not supported!\n"); merror=1;
        break;
        }
      else if (!strcasecmp(ctllineptr,"TITLE"))
        {
        if ((ctllineptr=strtok(NULL,"\n"))==NULL) continue;
        while (isspace(*ctllineptr) && *ctllineptr!='\0')
          ctllineptr++;
        strcpy(gtitle,ctllineptr);
        if (!quiet) printf("gtitle = %s\n",gtitle);
        }
      else if (!strcasecmp(ctllineptr,"UNDEF"))
        {
        if ((ctllineptr=strtok(NULL," \n"))==NULL)
          {
           merror=1; break;
          }
        misvalfound=sscanf(ctllineptr,"%f",&misval);
        if (!quiet) printf("misval = %g\n",misval);
        }
      else if (!strcasecmp(ctllineptr,"OPTIONS") ||
               !strcasecmp(ctllineptr,"FORMAT"))
        {
        if (!quiet) printf("options =");
        while ((ctllineptr=strtok(NULL," \n"))!=NULL)
          {
           if (!strcasecmp(ctllineptr,"SEQUENTIAL"))
             {
              if (!quiet) printf(" sequential");
             }
           else if (!strcasecmp(ctllineptr,"YREV"))
             {
              if (!quiet) printf(" yrev"); yrev=1;
             }
           else if (!strcasecmp(ctllineptr,"ZREV"))
             {
              if (!quiet) printf(" zrev"); zrev=1;
             }
           else if (!strcasecmp(ctllineptr,"TEMPLATE"))
             {
              if (binfile!='\0' && !strcmp(binfile,"%"))
                {
                 fprintf(stderr,"Multiple-file time series is not supported!\n");
                 merror=1; break;
                }
             }
           else if (!strcasecmp(ctllineptr,"BYTESWAPPED"))
             {
              if (!quiet) printf(" byteswapped"); byteswapped=1;
             }
           else if (!strcasecmp(ctllineptr,"BIG_ENDIAN") ||
                    !strcasecmp(ctllineptr,"CRAY_32BIT_IEEE"))
             {
              if (!quiet) printf(" bigendian"); bigendian=1;
             }
           else if (!strcasecmp(ctllineptr,"LITTLE_ENDIAN"))
             {
              if (!quiet) printf(" littleendian"); littleendian=1;
             }
          }
        if (!quiet) printf("\n");
        if (merror) break;
        }
      else if (!strcasecmp(ctllineptr,"SEQUENTIAL"))
        {
        if (!quiet) printf("sequential\n");
        }
      else if (!strcasecmp(ctllineptr,"YREV"))
        {
        if (!quiet) printf("yrev\n"); yrev=1;
        }
      else if (!strcasecmp(ctllineptr,"ZREV"))
        {
        if (!quiet) printf("zrev\n"); zrev=1;
        }
      else if (!strcasecmp(ctllineptr,"TEMPLATE"))
        {
        if (binfile!='\0' && !strcmp(binfile,"%"))
          {
           fprintf(stderr,"Multiple-file time series is not supported!\n");
           merror=1; break;
          }
        }
      else if (!strcasecmp(ctllineptr,"BYTESWAPPED"))
        {
        if (!quiet) printf("byteswapped\n"); byteswapped=1;
        }
      else if (!strcasecmp(ctllineptr,"BIG_ENDIAN") ||
               !strcasecmp(ctllineptr,"CRAY_32BIT_IEEE"))
        {
        if (!quiet) printf("bigendian\n"); bigendian=1;
        }
      else if (!strcasecmp(ctllineptr,"LITTLE_ENDIAN"))
        {
        if (!quiet) printf("littleendian\n"); littleendian=1;
        }
      else if (!strcasecmp(ctllineptr,"XDEF"))
        {
        if ((ctllineptr=strtok(NULL," \n"))==NULL)
```

```c
          {
           fprintf(stderr,"Missing the XDEF length!\n"); merror=1; break;
          }
        sscanf(ctllineptr,"%d",&xlen);
        if (xlen < 1)
          {
           fprintf(stderr,"XDEF length must be greater than 0!\n"); merror=1;
           break;
          }
        if ((xvals=(float *)malloc(sizeof(float)*xlen))==NULL)
          {
           fprintf(stderr,"Memory allocation error!\n"); merror=1; break;
          }
        if ((ctllineptr=strtok(NULL," \n"))==NULL)
          {
           fprintf(stderr,"Missing the XDEF mapping!\n"); merror=1; break;
          }
        if (!strcasecmp(ctllineptr,"LINEAR"))
          {
           if ((ctllineptr=strtok(NULL," \n"))==NULL)
             {
              fprintf(stderr,"Missing the linear XDEF starting value!\n");
              merror=1; break;
             }
           sscanf(ctllineptr,"%f",&xstart);
           if ((ctllineptr=strtok(NULL," \n"))==NULL)
             {
              fprintf(stderr,"Missing the linear XDEF increment value!\n");
              merror=1; break;
             }
           sscanf(ctllineptr,"%f",&xincr);
           for (x=0; x < xlen; x++)
             xvals[x]=xstart+x*xincr;
          }
        else if (!strcasecmp(ctllineptr,"LEVELS"))
          {
           x=0;
           while ((ctllineptr=strtok(NULL," \t\n"))!=NULL && x < xlen)
             if (sscanf(ctllineptr,"%f",&xval)) xvals[x++]=xval;
           if (x < xlen)
             while (fgets(ctlline,MAXLINELEN,ctlfile)!=NULL && !merror)
               {
                ctllineptr=strtok(ctlline," \t\n");
                while (ctllineptr!=NULL && x < xlen)
                  {
                   if (sscanf(ctllineptr,"%f",&xval)) yvals[x++]=xval;
                   else
                     {
                      merror=1; break;
                     }
                   ctllineptr=strtok(NULL," \t\n");
                  }
                if (x >= xlen) break;
               }
           if (x < xlen)
             {
              fprintf(stderr,"Only found %d XDEF levels, need %d!\n",x,xlen);
              merror=1; break;
             }
          }
        else
          {
           fprintf(stderr,"Unknown XDEF mapping!\n"); merror=1; break;
          }
        if (!quiet)
          {
           printf("xvals (%d) =",xlen);
           for (x=0; x < xlen; x++)
             printf(" %f",xvals[x]);
           printf("\n");
          }
       }
     else if (!strcasecmp(ctllineptr,"YDEF"))
       {
        if ((ctllineptr=strtok(NULL," \n"))==NULL)
          {
           fprintf(stderr,"Missing the YDEF length!\n"); merror=1; break;
          }
        sscanf(ctllineptr,"%d",&ylen);
        if (ylen < 1)
          {
           fprintf(stderr,"YDEF length must be greater than 0!\n"); merror=1;
           break;
          }
        if ((yvals=(float *)malloc(sizeof(float)*ylen))==NULL)
          {
           fprintf(stderr,"Memory allocation error!\n"); merror=1; break;
          }
        if ((ctllineptr=strtok(NULL," \n"))==NULL)
          {
           fprintf(stderr,"Missing the YDEF mapping!\n"); merror=1; break;
          }
        if (!strcasecmp(ctllineptr,"LINEAR"))
          {
           if ((ctllineptr=strtok(NULL," \n"))==NULL)
             {
              fprintf(stderr,"Missing the linear YDEF starting value!\n");
              merror=1; break;
             }
           sscanf(ctllineptr,"%f",&ystart);
           if ((ctllineptr=strtok(NULL," \n"))==NULL)
             {
              fprintf(stderr,"Missing the linear YDEF increment value!\n");
              merror=1; break;
             }
           sscanf(ctllineptr,"%f",&yincr);
           for (y=0; y < ylen; y++)
             yvals[y]=ystart+y*yincr;
          }
        else if (!strcasecmp(ctllineptr,"GAUSR15"))  /* 40 latitudes */
          {
           if ((ctllineptr=strtok(NULL," \n"))==NULL)
             {
              fprintf(stderr,"Missing the gaussian R15 YDEF starting value!\n");
              merror=1; break;
             }
           sscanf(ctllineptr,"%d",&gausystart);
           if (!gaussian_lats(ylen,gausystart,yvals,20))
             {
              merror=1; break;
             }
          }
        else if (!strcasecmp(ctllineptr,"GAUSR20"))  /* 51 latitudes */
          {
           if ((ctllineptr=strtok(NULL," \n"))==NULL)
             {
              fprintf(stderr,"Missing the gaussian R20 YDEF starting value!\n");
              merror=1; break;
             }
           sscanf(ctllineptr,"%d",&gausystart);
```

```
            if (!gaussian_lats(ylen,gausystart,yvals,26))
              {
               merror=1; break;
              }
           }
         else if (!strcasecmp(ctllineptr,"GAUSR30"))  /* 80 latitudes */
           {
            if ((ctllineptr=strtok(NULL," \n"))==NULL)
              {
               fprintf(stderr,"Missing the gaussian R30 YDEF starting value!\n");
               merror=1; break;
              }
            sscanf(ctllineptr,"%d",&gausystart);
            if (!gaussian_lats(ylen,gausystart,yvals,40))
              {
               merror=1; break;
              }
           }
         else if (!strcasecmp(ctllineptr,"GAUSR40"))  /* 102 latitudes */
           {
            if ((ctllineptr=strtok(NULL," \n"))==NULL)
              {
               fprintf(stderr,"Missing the gaussian R40 YDEF starting value!\n");
               merror=1; break;
              }
            sscanf(ctllineptr,"%d",&gausystart);
            if (!gaussian_lats(ylen,gausystart,yvals,51))
              {
               merror=1; break;
              }
           }
         else if (!strcasecmp(ctllineptr,"LEVELS"))
           {
            y=0;
            while ((ctllineptr=strtok(NULL," \t\n"))!=NULL && y < ylen)
              if (sscanf(ctllineptr,"%f",&yval)) yvals[y++]=yval;
            if (y < ylen)
              while (fgets(ctlline,MAXLINELEN,ctlfile)!=NULL && !merror)
                {
                 ctllineptr=strtok(ctlline," \t\n");
                 while (ctllineptr!=NULL && y < ylen)
                   {
                    if (sscanf(ctllineptr,"%f",&yval)) yvals[y++]=yval;
                    else
                      {
                       merror=1; break;
                      }
                    ctllineptr=strtok(NULL," \t\n");
                   }
                 if (y >= ylen) break;
                }
            if (y < ylen)
              {
               fprintf(stderr,"Only found %d YDEF levels, need %d!\n",y,ylen);
               merror=1; break;
              }
           }
         else
           {
            fprintf(stderr,"Unknown YDEF mapping!\n"); merror=1; break;
           }
         if (!quiet)
           {
            printf("yvals (%d) =",ylen);
            for (y=0; y < ylen; y++)
```

```
              printf(" %f",yvals[y]);
            printf("\n");
           }
        }
      else if (!strcasecmp(ctllineptr,"ZDEF"))
        {
         if ((ctllineptr=strtok(NULL," \n"))==NULL)
           {
            fprintf(stderr,"Missing the ZDEF length!\n"); merror=1; break;
           }
         sscanf(ctllineptr,"%d",&zlen);
         if (zlen < 1)
           {
            fprintf(stderr,"ZDEF length must be greater than 0!\n"); merror=1;
            break;
           }
         if ((zvals=(float *)malloc(sizeof(float)*zlen))==NULL)
           {
            fprintf(stderr,"Memory allocation error!\n"); merror=1; break;
           }
         if ((ctllineptr=strtok(NULL," \n"))==NULL)
           {
            fprintf(stderr,"Missing the ZDEF mapping!\n"); merror=1; break;
           }
         if (!strcasecmp(ctllineptr,"LINEAR"))
           {
            if ((ctllineptr=strtok(NULL," \n"))==NULL)
              {
               fprintf(stderr,"Missing the linear ZDEF starting value!\n");
               merror=1; break;
              }
            sscanf(ctllineptr,"%f",&zstart);
            if ((ctllineptr=strtok(NULL," \n"))==NULL)
              {
               fprintf(stderr,"Missing the linear ZDEF increment value!\n");
               merror=1; break;
              }
            sscanf(ctllineptr,"%f",&zincr);
            for (z=0; z < zlen; z++)
              zvals[z]=zstart+z*zincr;
           }
         else if (!strcasecmp(ctllineptr,"LEVELS"))
           {
            z=0;
            while ((ctllineptr=strtok(NULL," \n"))!=NULL && z < zlen)
              if (sscanf(ctllineptr,"%f",&zval)) zvals[z++]=zval;
            if (z < zlen)
              while (fgets(ctlline,MAXLINELEN,ctlfile)!=NULL && !merror)
                {
                 ctllineptr=strtok(ctlline," \t\n");
                 while (ctllineptr!=NULL && z < zlen)
                   {
                    if (sscanf(ctllineptr,"%f",&zval)) zvals[z++]=zval;
                    else
                      {
                       merror=1; break;
                      }
                    ctllineptr=strtok(NULL," \t\n");
                   }
                 if (z >= zlen) break;
                }
            if (z < zlen)
              {
               fprintf(stderr,"Only found %d ZDEF levels, need %d!\n",z,zlen);
               merror=1; break;
```

```c
              }
            }
          else
            {
             fprintf(stderr,"Unknown ZDEF mapping!\n"); merror=1; break;
            }
          if (!quiet)
            {
             printf("zvals (%d) =",zlen);
             for (z=0; z < zlen; z++)
               printf(" %f",zvals[z]);
             printf("\n");
            }
        }
      else if (!strcasecmp(ctllineptr,"TDEF"))
        {
          if ((ctllineptr=strtok(NULL," \n"))==NULL)
            {
             fprintf(stderr,"Missing the TDEF length!\n"); merror=1; break;
            }
          sscanf(ctllineptr,"%d",&tlen);
          if (tlen < 1)
            {
             fprintf(stderr,"TDEF length must be greater than 0!\n"); merror=1;
             break;
            }
          if ((tvals=(float *)malloc(sizeof(float)*tlen))==NULL)
            {
             fprintf(stderr,"Memory allocation error!\n"); merror=1; break;
            }
          if ((ctllineptr=strtok(NULL," \n"))==NULL)
            {
             fprintf(stderr,"Missing the TDEF mapping!\n"); merror=1; break;
            }
          if (!strcasecmp(ctllineptr,"LINEAR"))
            {
             if ((ctllineptr=strtok(NULL," \n"))==NULL)
               {
                fprintf(stderr,"Missing the linear TDEF starting value!\n");
                merror=1; break;
               }
             sscanf(ctllineptr,"%s",tstart);
             if ((ctllineptr=strtok(NULL," \n"))==NULL)
               {
                fprintf(stderr,"Missing the linear TDEF increment value!\n");
                merror=1; break;
               }
             sscanf(ctllineptr,"%s",tincr);
             hour=0; minute=0; day=1;
             switch (strlen(tstart))
               {
                case 15:  /* hh:mmZddmmmyyyy */
                  sscanf(tstart,"%d:%dZ%d%3s%d",&hour,&minute,&day,month,&year);
                  break;
                case 14:  /* hh:mmZdmmmyyyy */
                  sscanf(tstart,"%d:%dZ%d%3s%d",&hour,&minute,&day,month,&year);
                  break;
                case 13:  /* hh:mmZddmmmyy | hh:mmZmmmyyyy | hh:Zddmmmyyyy */
                  if (isdigit(tstart[6]))
                    sscanf(tstart,"%d:%dZ%d%3s%d",&hour,&minute,&day,month,
                           &year);
                  else if (toupper(tstart[5])=='Z')
                    sscanf(tstart,"%d:%dZ%3s%d",&hour,&minute,month,&year);
                  else
                    sscanf(tstart,"%d:Z%d%3s%d",&hour,&day,month,&year);
                  break;
                case 12:  /* hh:mmZdmmmyy | hh:Zdmmmyyyy | hhZddmmmyyyy */
                  if (tstart[2]==':')
                    {
                     if (toupper(tstart[5])=='Z')
                       sscanf(tstart,"%d:%dZ%d%3s%d",&hour,&minute,&day,month,
                              &year);
                     else
                       sscanf(tstart,"%d:Z%d%3s%d",&hour,&day,month,&year);
                    }
                  else
                    sscanf(tstart,"%dZ%d%3s%d",&hour,&day,month,&year);
                  break;
                case 11:  /* hh:Zddmmmyy | hh:mmZmmmyy | hh:Zmmmyyyy | */
                          /* hhZdmmmyyyy                              */
                  if (tstart[2]==':')
                    {
                     if (isdigit(tstart[5]))
                       sscanf(tstart,"%d:Z%d%3s%d",&hour,&day,month,&year);
                     else if (toupper(tstart[5])=='Z')
                       sscanf(tstart,"%d:%dZ%3s%d",&hour,&minute,month,&year);
                     else
                       sscanf(tstart,"%d:Z%3s%d",&hour,month,&year);
                    }
                  else
                    sscanf(tstart,"%dZ%d%3s%d",&hour,&day,month,&year);
                  break;
                case 10:  /* hh:Zdmmmyy | hhZddmmmyy | hhZmmmyyyy */
                  if (tstart[2]==':')
                    sscanf(tstart,"%d:Z%d%3s%d",&hour,&day,month,&year);
                  else if (isdigit(tstart[3]))
                    sscanf(tstart,"%dZ%d%3s%d",&hour,&day,month,&year);
                  else
                    sscanf(tstart,"%dZ%3s%d",&hour,month,&year);
                  break;
                case 9:  /* hhZdmmmyy | hh:Zmmmyy | ddmmmyyyy */
                  if (toupper(tstart[2])=='Z')
                    sscanf(tstart,"%dZ%d%3s%d",&hour,&day,month,&year);
                  else if (tstart[2]==':')
                    sscanf(tstart,"%d:Z%3s%d",&hour,month,&year);
                  else
                    sscanf(tstart,"%d%3s%d",&day,month,&year);
                  break;
                case 8:  /* dmmmyyyy | hhZmmmyy */
                  if (toupper(tstart[2])!='Z')
                    sscanf(tstart,"%d%3s%d",&day,month,&year);
                  else
                    sscanf(tstart,"%dZ%3s%d",&hour,month,&year);
                  break;
                case 7:  /* ddmmmyy | mmmyyyy */
                  if (isdigit(tstart[0]))
                    sscanf(tstart,"%d%3s%d",&day,month,&year);
                  else
                    sscanf(tstart,"%3s%d",month,&year);
                  break;
                case 6:  /* dmmmyy */
                  sscanf(tstart,"%d%3s%d",&day,month,&year);
                  break;
                case 5:  /* mmmyy */
                  sscanf(tstart,"%3s%d",month,&year);
                  break;
                default:
                  fprintf(stderr,"Invalid TDEF starting time \"%s\"!\n",
                          tstart);
                  merror=1; break;
```

```
              }
            if (year < 100)
              {
               if (year > 49) year+=1900;
               else year+=2000;
              }
            if (!merror)
              {
               for (t=0; t < 12; t++)
                 if (!strcasecmp(month,months[t])) break;
               if (t==12)
                 {
                  fprintf(stderr,"Invalid TDEF starting time \"%s\"!\n",
                          tstart);
                  merror=1; break;
                 }
               else
                 {
                  sprintf(tstart2,"%d-%02d-%02d %02d:%02d:00.0",year,t+1,day,
                          hour,minute);
                  sscanf(tincr,"%d%s",&tincr1,tincr2);
                  if (!strcasecmp(tincr2,"mn"))
                    sprintf(tincr2,"minutes");
                  else if (!strcasecmp(tincr2,"hr"))
                    sprintf(tincr2,"hours");
                  else if (!strcasecmp(tincr2,"dy"))
                    sprintf(tincr2,"days");
                  else if (!strcasecmp(tincr2,"mo"))
                    sprintf(tincr2,"months");
                  else if (!strcasecmp(tincr2,"yr"))
                    sprintf(tincr2,"years");
                  else
                    {
                     fprintf(stderr,"Invalid TDEF time increment \"%s\"!\n",
                             tincr);
                     merror=1; break;
                    }
                  sprintf(tstart3,"%s since %s",tincr2,tstart2);
                 }
              }
            if (!quiet) printf("time = %s\n",tstart3);
            for (t=0; t < tlen; t++)
              tvals[t]=t*tincr1;
           }
         else
           {
            fprintf(stderr,"Unknown TDEF mapping!\n"); merror=1; break;
           }
         if (!quiet)
           {
            printf("tvals (%d) =",tlen);
            for (t=0; t < tlen; t++)
              printf(" %f",tvals[t]);
            printf("\n");
           }
        }
      else if (!strcasecmp(ctllineptr,"VARS"))
        {
         if ((ctllineptr=strtok(NULL," \n"))==NULL)
           {
            fprintf(stderr,"Missing the number of variables!\n"); merror=1;
            break;
           }
         sscanf(ctllineptr,"%d",&nvars);
         if (!quiet) printf("nvars = %d\n",nvars);
```

```
        if ((varnames=(char *)malloc(128*nvars))==NULL)
          {
           fprintf(stderr,"Memory allocation error!\n"); merror=1; break;
          }
        for (c=0; c < 128*nvars; c++)
          varnames[c]=' ';
        if ((vardescs=(char *)malloc(nvars*MAXSTRLEN))==NULL)
          {
           fprintf(stderr,"Memory allocation error!\n"); merror=1; break;
          }
        for (c=0; c < nvars*MAXSTRLEN; c++)
          vardescs[c]=' ';
        if ((varzlens=(int *)malloc(sizeof(int)*nvars))==NULL)
          {
           fprintf(stderr,"Memory allocation error!\n"); merror=1; break;
          }
        if ((varzdims=(int *)malloc(sizeof(int)*nvars))==NULL)
          {
           fprintf(stderr,"Memory allocation error!\n"); merror=1; break;
          }
        if ((dimzlens=(int *)malloc(sizeof(int)*nvars))==NULL)
          {
           fprintf(stderr,"Memory allocation error!\n"); merror=1; break;
          }
        for (v=0; v < nvars; v++)
          {
           varzlens[v]=0; varzdims[v]=(-1);
          }
        for (v=0; v < nvars; v++)
          {
           if (fgets(varrec,MAXSTRLEN,ctlfile)==NULL)
             {
              fprintf(stderr,"Only found %d variables, need %d!\n",v,nvars);
              merror=1; break;
             }
           varrecptr=strtok(varrec," \n");
           if (varrecptr==NULL || !strcasecmp(varrecptr,"ENDVARS"))
             {
              fprintf(stderr,"Only found %d variables, need %d!\n",v,nvars);
              merror=1; break;
             }
           varnamesptr=varnames+v*128; strcpy(varnamesptr,varrecptr);
           if ((varrecptr=strtok(NULL," \n"))==NULL)
             {
              fprintf(stderr,"Missing the numer of levels for variable %d (%s)!\n",
                      v+1,varnamesptr);
              merror=1; break;
             }
           sscanf(varrecptr,"%d",&varzlen);
           if (varzlen < 0 || varzlen > zlen)
             {
              fprintf(stderr,"Invalid number of levels (%d) for variable %d (%s)!\n",
                      varzlen,v+1,varnamesptr);
              merror=1; break;
             }
           if ((varrecptr=strtok(NULL," \n"))!=NULL)
             sscanf(varrecptr,"%s",gribunits);
           else strcpy(gribunits,"99");
           if (strcmp(gribunits,"99"))
             {
              fprintf(stderr,"Warning: GRIB processing or specialized handling of the/
variables is not\n");
              fprintf(stderr,"         supported!  The units value should be 99.\n");
             }
           vardescsptr=vardescs+v*MAXSTRLEN;
```

```c
            if ((varrecptr=strtok(NULL,"\n"))!=NULL)
              {
               while (isspace(*varrecptr) && *varrecptr!='\0')
                 varrecptr++;
               strcpy(vardescsptr,varrecptr);
              }
            else strcpy(vardescsptr,"None");
            if (varzlen!=0)
              {
               zmatch=1;
               for (z=0; z < nvars; z++)
                 if (varzlen==varzlens[z])
                   {
                    zmatch=0; break;
                   }
               if (zmatch)
                 {
                  nzdims++; dimzlens[d++]=varzlen;
                 }
               for (z=0; z < nzdims; z++)
                 if (dimzlens[z]==varzlen)
                   varzdims[v]=z+1;
              }
            varzlens[v]=varzlen;
            if (!quiet)
              printf("%s %d %s %s\n",varnamesptr,varzlen,gribunits,
                     vardescsptr);
           }
        }
      else if (!strcasecmp(ctllineptr,"ENDVARS"))
        ctlfdone=1;  /* End of GrADS control file */
      if (merror) break;
     }
   if (binfile[0]=='\0')
     {
      fprintf(stderr,"\nMissing the IEEE data file!\n"); merror=1;
     }
   if (stat(binfile,&binfileinfo)!=0)
     {
      fprintf(stderr,"\nCannot read the IEEE data file!\n"); merror=1;
     }
#ifdef HOST_T90
   if (byteswapped)
     {
      fprintf(stderr,"\nByteswapping is not supported on Cray machines!\n");
      merror=1;
     }
#endif
   if (bigendian && littleendian)
     {
      fprintf(stderr,"Cannot have both \"BIG_ENDIAN\" and \"LITTLE_ENDIAN\" defined!\n
");
      merror=1;
     }
   else
     {
      if (endian==MSB)
        {
         if (bigendian) byteswapped=0;
         if (littleendian) byteswapped=1;
        }
      else  /* endian==LSB */
        {
         if (littleendian) byteswapped=0;
         if (bigendian) byteswapped=1;
        }
     }
   if (!misvalfound)
     {
      fprintf(stderr,"\nNo undefined (missing) value is specified!\n");
      merror=1;
     }
   if (!quiet)
     {
      if (merror) printf("\nParsing aborted!\n----------------\n\n");
      else printf("\nDone parsing.\n-------------\n\n");
     }

   /* Write the netCDF file using the control file's information */
   if (!merror)
     {
      for (c=0; c < 128; c++)
        ncfile[c]=' ';
      strcpy(ncfile,argv[2]);
      if ((valuesxy=(float *)malloc(sizeof(float)*xlen*ylen))==NULL)
        {
         fprintf(stderr,"Memory allocation error!\n"); merror=1;
        }
      if ((varranges=(float *)malloc(sizeof(float)*2*nvars))==NULL)
        {
         fprintf(stderr,"Memory allocation error!\n"); merror=1;
        }
#define MAX(a,b) ((a) > (b) ? (a) : (b))
      mxdimout=MAX(MAX(xlen,ylen),zlen); ndims=nzdims+3;
      if (!merror)
        if ((dimvals=(float *)malloc(sizeof(float)*mxdimout*ndims))==NULL)
          {
           fprintf(stderr,"Memory allocation error!\n"); merror=1;
          }
      if (!merror)
        if ((lendims=(int *)malloc(sizeof(int)*ndims))==NULL)
          {
           fprintf(stderr,"Memory allocation error!\n"); merror=1;
          }
      if (!merror)
        if ((cdimnam=(char *)malloc(128*ndims))==NULL)
          {
           fprintf(stderr,"Memory allocation error!\n"); merror=1;
          }
      if (!merror)
        if ((cdimlnam=(char *)malloc(MAXSTRLEN*ndims))==NULL)
          {
           fprintf(stderr,"Memory allocation error!\n"); merror=1;
          }
      if (!merror)
        if ((cdimunits=(char *)malloc(MAXSTRLEN*ndims))==NULL)
          {
           fprintf(stderr,"Memory allocation error!\n"); merror=1;
          }
      if (!merror)
        if ((cart_axis=(char *)malloc(ndims))==NULL)
          {
           fprintf(stderr,"Memory allocation error!\n"); merror=1;
          }
      if (!merror)
        if ((ipositive=(int *)malloc(sizeof(int)*ndims))==NULL)
          {
           fprintf(stderr,"Memory allocation error!\n"); merror=1;
          }
      if (!merror)
```

```c
        if ((cvarnam=(char *)malloc(128*nvars))==NULL)
          {
           fprintf(stderr,"Memory allocation error!\n"); merror=1;
          }
      if (!merror)
        if ((cvarlnam=(char *)malloc(MAXSTRLEN*nvars))==NULL)
          {
           fprintf(stderr,"Memory allocation error!\n"); merror=1;
          }
      if (!merror)
        if ((cvarunits=(char *)malloc(MAXSTRLEN*nvars))==NULL)
          {
           fprintf(stderr,"Memory allocation error!\n"); merror=1;
          }
      if (!merror)
        if ((nbits=(int *)malloc(sizeof(int)*nvars))==NULL)
          {
           fprintf(stderr,"Memory allocation error!\n"); merror=1;
          }
      if (!merror)
        if ((idimindx=(int *)malloc(sizeof(int)*ndims*nvars))==NULL)
          {
           fprintf(stderr,"Memory allocation error!\n"); merror=1;
          }
      if (!merror)
        if ((nvdims=(int *)malloc(sizeof(int)*nvars))==NULL)
          {
           fprintf(stderr,"Memory allocation error!\n"); merror=1;
          }
      if (!merror)
        if ((maxnitems=(int *)malloc(sizeof(int)*nvars))==NULL)
          {
           fprintf(stderr,"Memory allocation error!\n"); merror=1;
          }
      if (!merror)
        if ((validr=(float *)malloc(sizeof(float)*2*nvars))==NULL)
          {
           fprintf(stderr,"Memory allocation error!\n"); merror=1;
          }
      if (!merror)
        if ((vspval=(float *)malloc(sizeof(float)*nvars))==NULL)
          {
           fprintf(stderr,"Memory allocation error!\n"); merror=1;
          }
      if (!merror)
        if ((lspval=(int *)malloc(sizeof(int)*nvars))==NULL)
          {
           fprintf(stderr,"Memory allocation error!\n"); merror=1;
          }
      if (!merror)
        if ((istart=(int *)malloc(sizeof(int)*ndims))==NULL)
          {
           fprintf(stderr,"Memory allocation error!\n"); merror=1;
          }
      if (!merror)
        if ((icount=(int *)malloc(sizeof(int)*ndims))==NULL)
          {
           fprintf(stderr,"Memory allocation error!\n"); merror=1;
          }
      if (!merror)
#ifdef HOST_T90
        WRITE_GRADS2NC(_cptofcd(binfile,strlen(binfile)),
                       _cptofcd(ncfile,strlen(ncfile)),
                       _cptofcd(gtitle,strlen(gtitle)),
                       &misval,&xlen,&ylen,&zlen,&tlen,xvals,yvals,zvals,tvals,
                       &nzdims,dimzlens,varzdims,&nvars,varzlens,
                       _cptofcd(varnames,nvars*128),
                       _cptofcd(vardescs,nvars*MAXSTRLEN),
                       valuesxy,varranges,&byteswapped,&yrev,&zrev,
                       _cptofcd(tstart3,strlen(tstart3)),&calcrange,&verbose,
                       &quiet,&mxdimout,&ndims,dimvals,lendims,
                       _cptofcd(cdimnam,128*ndims),
                       _cptofcd(cdimlnam,MAXSTRLEN*ndims),
                       _cptofcd(cdimunits,MAXSTRLEN*ndims),
                       _cptofcd(cart_axis,ndims),ipositive,
                       _cptofcd(cvarnam,128*nvars),
                       _cptofcd(cvarlnam,MAXSTRLEN*nvars),
                       _cptofcd(cvarunits,MAXSTRLEN*nvars),nbits,idimindx,
                       nvdims,maxnitems,validr,vspval,_btol(lspval),istart,
                       icount);
#else
        write_grads2nc_(binfile,ncfile,gtitle,&misval,&xlen,&ylen,&zlen,&tlen,
                        xvals,yvals,zvals,tvals,&nzdims,dimzlens,varzdims,
                        &nvars,varzlens,varnames,vardescs,valuesxy,varranges,
                        &byteswapped,&yrev,&zrev,tstart3,&calcrange,&verbose,
                        &quiet,&mxdimout,&ndims,dimvals,lendims,cdimnam,
                        cdimlnam,cdimunits,cart_axis,ipositive,cvarnam,
                        cvarlnam,cvarunits,nbits,idimindx,nvdims,maxnitems,
                        validr,vspval,lspval,istart,icount);
#endif
     }

   /* Clean up */
   if (xvals!=NULL) free(xvals);
   if (yvals!=NULL) free(yvals);
   if (zvals!=NULL) free(zvals);
   if (tvals!=NULL) free(tvals);
   if (varnames!=NULL) free(varnames);
   if (vardescs!=NULL) free(vardescs);
   if (varzlens!=NULL) free(varzlens);
   if (varzdims!=NULL) free(varzdims);
   if (dimzlens!=NULL) free(dimzlens);
   if (valuesxy!=NULL) free(valuesxy);
   if (varranges!=NULL) free(varranges);
   if (dimvals!=NULL) free(dimvals);
   if (lendims!=NULL) free(lendims);
   if (cdimnam!=NULL) free(cdimnam);
   if (cdimlnam!=NULL) free(cdimlnam);
   if (cdimunits!=NULL) free(cdimunits);
   if (cart_axis!=NULL) free(cart_axis);
   if (ipositive!=NULL) free(ipositive);
   if (cvarnam!=NULL) free(cvarnam);
   if (cvarlnam!=NULL) free(cvarlnam);
   if (cvarunits!=NULL) free(cvarunits);
   if (nbits!=NULL) free(nbits);
   if (idimindx!=NULL) free(idimindx);
   if (nvdims!=NULL) free(nvdims);
   if (maxnitems!=NULL) free(maxnitems);
   if (validr!=NULL) free(validr);
   if (vspval!=NULL) free(vspval);
   if (lspval!=NULL) free(lspval);
   if (istart!=NULL) free(istart);
   if (icount!=NULL) free(icount);
   fclose(ctlfile);
   if (merror) exit(1);
   else exit(0);
  }


/* Calculate the Gaussian latitudes */
```

```
/* Note: This code was adapted from a program originally in      */
/*        "Numerical Recipes" (Cambridge University Press, 1989) */
int gaussian_lats(int l, int s, float *yvals, int m)
  {
   double z, z1, p1, p2, p3, pp, eps=3.0E-14;
   int n=m*2, i, j;
   float *lats;
#define M_180DIVPI 57.29577951308

   if ((lats=(float *)malloc(sizeof(float)*n))==NULL)
     {
      fprintf(stderr,"Memory allocation error!\n"); return(0);
     }
   for (i=1; i <= m; i++)
     {
      z=cos(M_PI*(i-.25)/(n+.5));
label1:
      p1=1.0; p2=0.0;
      for (j=1; j <= n; j++)
        {
         p3=p2; p2=p1; p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
        }
      pp=n*(z*p1-p2)/(z*z-1.0); z1=z; z=z1-p1/pp;
      if (fabs(z-z1) > eps) goto label1;
      lats[i-1]=(float)z*(-1);
      lats[n-i]=(float)z;
     }
   s--; l--; j=0;
   for (i=s; i <= l; i++)
     {
      yvals[j++]=(float)asin(lats[i])*M_180DIVPI;
     }
   free(lats); return(1);
  }


/* Swap the bytes in a 32-bit float (MSB<->LSB) */
#ifdef HOST_T90
int BYTE_SWAP(float *swaparray, int *xlen, int *ylen)
#else
int byte_swap_(float *swaparray, int *xlen, int *ylen)
#endif
  {
   float *swaparrayptr=swaparray;
   int xy, xylen=(*xlen)*(*ylen);
   unsigned char *byteptr, byteval;

   for (xy=0; xy < xylen; xy++)
     {
      byteptr=(unsigned char *)swaparrayptr;
      byteval=byteptr[0]; byteptr[0]=byteptr[3]; byteptr[3]=byteval;
      byteval=byteptr[1]; byteptr[1]=byteptr[2]; byteptr[2]=byteval;
      swaparrayptr++;
     }
   return(0);
  }
```