平成 15 年 12 月 19 日
気象庁　気候・海洋気象部

配信資料に関する技術情報（気象編）第１５４号

～暖・寒候期予報 GPV 等のサンプルデータの提供について～

配信資料に関する技術情報（気象編）第150号（平成15年12月8日発表）により、暖・寒候期予報GPV等のFTP方式による提供開始及びそのフォーマットについてお知らせしました。これに関連し、本日、（財）気象業務支援センターにサンプルデータを提供しますのでお知らせします。

１．サンプルデータを収録したCD-ROMの構成について

フォルダ名と収録ファイルは次のとおりです。

MGPV	メンバー別全球格子点値（40 ファイル）及び同 tar 結合ファイル
GPV	アンサンブル統計全球格子点値（29 ファイル）及び同 tar 結合ファイル
GDCs	ガイダンス（暖候期３ファイル）及び同 tar 結合ファイル
GDCw	ガイダンス（寒候期３ファイル）及び同 tar 結合ファイル
OCNCCAs	統計予測資料（暖候期６ファイル）及び同 tar 結合ファイル
OCNCCAw	統計予測資料（寒候期６ファイル）及び同 tar 結合ファイル

２．解読（デコード）処理について

暖・寒候期予報アンサンブル格子点値（メンバー別全球格子点値（１項の“MGPV”に収録）及びアンサンブル統計全球格子点値（同“GPV”））は解読処理が必要です。

解読用サンプルプログラムおよびその使用方法を解説を添付しておりますので参考にしてください。

なお、暖・寒候期予報支援資料（ガイダンス及び統計予測資料）は、CSV形式（カンマで区切られたテキストデータ）のため、解読処理は必要ありません。

平 成　１５　年　１２　月
気象庁　気候・海洋気象部

暖・寒候期予報アンサンブル格子点値ファイルの解読（デコード）処理について

１．解読サンプルプログラムのソースコード
別紙参照

２．利用方法
以下、解読サンプルプログラムの
ソースコードのファイル名：dcd_6megrib2_sample.c
実行ファイル名　　：dcd_6merib2
とする。

（１）ccコマンドによりコンパイルしてください。その際標準算術関数を利用可能なようにライブラリをリンクしてください。
実行例）　$ cc dcd_6megrib2_sample.c - o dcd_6merib2 -lm
（dcd_6megrib2 と言う実行ファイルが生成される）

- ANSI 準拠の c コンパイラでコンパイルできます。UNIX (HP-UX, HI-UX/WE2) 及び Linux (RedHat) での動作を確認しています。
- リトルエンディアンマシンにも対応しています。

（２）次のコマンドを入力することにより、暖・寒候期予報アンサンブル格子点値，各節の内容が端末に表示されると共に、４バイト実数形式でデコードされたデータがファイルに書き出される。
実行例）$ dcd_6megrib2 {暖・寒候期予報アンサンブル格子点値ファイル名}

- デコードされたデータは、予報時間ごとに、サンプルプログラムで割り付けた複数のファイルに出力されます。ファイル名は227行で使用されている関数identifydata 994～1006行で割り付けています。
- ファイルの出力を止めたい場合は、サンプルプログラム83行目の
  #define IFOUT 1
  を
  #define IFOUT 0
  など、1以外の数字に変更してください。

別紙

解読用サンプルプログラムのソースコード
（暖・寒候期予報アンサンブル格子点値ファイル用）

```
/*****************************************************************************
**       Sample Program to convert   6 month ensemble forecast GPV of JMA
**                                                  2003.11     JMA/CPD
**
**              Tested on Windows (gcc ,borlandc), Linux (gcc)
**                      and HP-UX (native c compiler)
**
*****************************************************************************/
/*     Description of grobal parameter
*     fl[30]                    :output filename used in out_data()
*     flg[30]                   :output file name used in out_grads() GrADS formatted file
*     flc[30]                   :output file name used in out_grads() GrADS control file
*     fla[30]                   :output file naem used in out_ascii()
*     *fp,*fpg,*fpc,*fpa   : file pointer corresponding to fl,flg,flc,fla
*     *fname                     : pointer to input file name
*     buffer[SIZEOFBUF]     :buffer to read grib parameter
*     map[SIZEOFBITBUF]     :buffer to read bitmap
*     buffer2[MAX_BUFF]     :buffer to read GPV data
*     recno                     :indicator of number of data
*     < section 0 >
*     len                       :Total length of GRIB
*     mtn                       :GRIB Master Table Number
*     < section 1 >
*     len1                      :Length of Section 1
*     iyyyy,imm,iday            :Reference Time
*     ltvn                      :GRIB Local Tables Version Number
*     < section 2>
*     len2                      :Length of Section 2
*     ns2                       :Number of Section (2 is assumed)
*     < section 3 >
*     len3                      :Length of section 3
*     ns3                       :Number of section (3 is assumed)
*     ijmx                      :Number of data points
*     Ngdt                      :Grid Definition Template Number
*     Ni                        :number of points along a parallel
*     Nj                        :number of points along a meridian
*     Di                        :i direction increment
*     Dj                        :j direction increment
*     La1                       :latitude of first grid point
*     La2                       :latitude of last grid point
*     Lo1                       :longitude of first grid point
*     Lo2                       :longitude of last grid point
*     < section 4 >
*     len4                      :Length of section 4
*     ns4                       :Number of section (4 is assumed)
*     pc                        :Parameter category
*     pn                        :Parameter number
*     vl1                       :value of first fixed surface
```

```
 49  *      dfn                    :Derived forecast (see Code Table 4.7)
 50  *      kt                     :Forecast time in units defined by octet 18
 51  *      lt                     :Length of the time range
 52  *      tys                    :Type of first fixed surface
 53  *      Npdt                   :Product Definition Template Number
 54  *      nmem                   :Perturbation number
 55  *      tyens                  :Type of ensemble forecast
 56  *      < section 5 >
 57  *      len5                   :Length of section 5
 58  *      ns5                    :Number of section
 59  *      ijdim                  :Number of data points
 60  *      nbit                   :Number of bits .....
 61  *      ntemplate              :Data Representation Template Number
 62  *      E                      :Binary scale factor
 63  *      D                      :Decimal scale factor
 64  *      R                      :Reference value
 65  *      < section 6 >
 66  *      len6                   :Length of section 6
 67  *      ns6                    :Number of section (6 is assumed)
 68  *      ifbitmap               :Bit-map indicator
 69  *      < section 7 >
 70  *      len7                   :Length of section 7
 71  *      ns7                    :Number of section (7 is assumed)
 72  *      nb                     :parameter of byte number
 73  *      mb                     :parameter of bit in octet
 74  *      data[MAX_IJX]          :Grid Point Value
 75  *      tcount,tcounta         :counter used in out_grads()
 76  ********************************************************************************/
 77  #include <stdio.h>
 78  #include <string.h>
 79  #include <math.h>
 80  #include <stdlib.h>
 81  #include <ctype.h>
 82
 83  #define IFOUT           1          /*   1 :Output decoded data                */
 84  #define MASK7           0x7F
 85  #define MASK1           0x80
 86  #define UNDEF          -19999.
 87  #define SIZEOFBUF      10000
 88  #define SIZEOFBITBUF 50000
 89  #define MAX_BUFF       50000      /* buffer size for gpv MAX_BUFF < 4294967295   2^32 */
 90  #define MAX_IJX        20000
 91      static    char          fl[30];
 92      char                    flg[30],flc[30],fla[30];
 93      FILE                    *fp,*fpg,*fpc,*fpa;
 94      char                    *fname;
 95      static unsigned char    buffer[SIZEOFBUF],map[SIZEOFBITBUF],buffer2[MAX_BUFF];
 96
 97      unsigned long           len,mtn;
 98      unsigned long           len1,iyyyy,imm,iday,ltvn;
 99      int                     recno;
100      unsigned long           len2,ns2;
101      unsigned long           len3,ns3,ijmx,Ngdt,Ni,Nj,Di,Dj;
```

```
102         long                    La1,La2,Lo1,Lo2;
103         unsigned long           len4,ns4,pc,pn,vl1,dfn,kt,lt,tys,Npdt;
104         long                    nmem,tyens;
105         unsigned long           len5,ns5,ijdim,nbit,ntemplate;
106         long                    E,D;
107         float                   R;
108         unsigned long           len6,ns6,ifbitmap;
109         unsigned long           len7,ns7,nb,mb;
110         static    float         data[MAX_IJX];
111         unsigned long           tcount=0,tcounta=0;
112     /*==========================================================================
113     */
114     /* TYPE DEF                                                                */
115     /*==========================================================================
116     */
117     /*--------------------------------------------------------------------------*/
118     /* TOOLS                                                                   */
119     /*--------------------------------------------------------------------------*/
120     void usage(void);
121     long longbybit(unsigned char *buf,unsigned long *pos,unsigned long nbit);
122     long convlong( unsigned char *string, int nbyte );
123     float convfloat( unsigned char *string, int nbyte );
124     long convlatlon(unsigned char *string );
125
126     /*--------------------------------------------------------------------------*/
127     /*      used in identifydata                                               */
128     /*--------------------------------------------------------------------------*/
129     void fproduct(void);
130     void flevel(void);
131     void fderived(void);
132     void fhemispher(void);
133     void fperiod(void);
134
135     /*--------------------------------------------------------------------------*/
136     /*    Identify Data (FILE NAME)                                            */
137     /*--------------------------------------------------------------------------*/
138     void identifydata(void);
139
140     /*--------------------------------------------------------------------------*/
141     /*    DECODE SECTION                                                       */
142     /*--------------------------------------------------------------------------*/
143     void dcd_sect0(void);
144     void dcd_sect1(void);
145     void dcd_sect2(void);
146     void dcd_sect3(void);
147     void dcd_sect4(void);
148     void dcd_sect5(void);
149     void dcd_sect6(void);
150     void dcd_sect7(void);
151
152     void out_data(void);
153
154     /*==========================================================================
```

```
155   */
156   /* MAIN PROGRAM                                                                    */
157   /*================================================================================*/
158   */
159   main(argc,argv)
160   int       argc;
161   char      **argv;
162   {
163       unsigned long      ns;
164       unsigned long      wkl;
165       char               wk[105],cns[5];
166           /*    Input File    */
167           if(2 != argc)
168                   {
169                   usage();
170                   exit(0);
171                   }
172           if(NULL == (fp = fopen(argv[1],"rb")))
173                   {
174                   printf("¥nCannot open FILE : %s¥n",argv[1]);
175                   usage();
176                   exit(1);
177                   }
178
179           fname=argv[1];
180           recno=0;
181   /*-------------------------------------------------------------------------*/
182   /*    DECODE SECT 0   : Indicator Section                                  */
183   /*-------------------------------------------------------------------------*/
184        dcd_sect0();
185   /*-------------------------------------------------------------------------*/
186   /*    DECODE SECT 1   : Identification Section                             */
187   /*-------------------------------------------------------------------------*/
188       printf("¥n<<SECTION 1>> : Identification Section ¥n");
189       dcd_sect1();
190   /*-------------------------------------------------------------------------*/
191   /*    DECODE SECT 2    : (Local Use Section)                               */
192   /*-------------------------------------------------------------------------*/
193       if( fread(buffer,sizeof(char), 5,fp) != 5 ){
194           printf("Read ERR SECT 2 !! File(%s)¥n",fname);
195           exit(72);
196           }
197       len2=convlong(&buffer[0], 4);
198       ns2=convlong(&buffer[4], 1);
199       fseek(fp,-5,1);              /*   Back Space 5 bytes   */
200           switch(ns2){
201               case 2:
202                   printf("<<SECTION 2>> : (Local Use Section)¥n");
203                   goto SECT2;
204               case 3:
205                   printf("¥n<<SECTION 3>> : Grid Definition Section¥n");
206                   goto SECT3;
207               default:
```

```
208                     printf("ERROR in Section 8 !!");
209                     printf(" len ns = %d %d¥n",wkl,buffer[0]);
210                     exit(99);
211             }
212     SECT2:
213         dcd_sect2();
214     /*-------------------------------------------------------------------------*/
215     /*     DECODE SECT 3        : Grid Definition Section                      */
216     /*-------------------------------------------------------------------------*/
217         printf("¥n<<SECTION 3>> : Grid Definition Section¥n");
218     SECT3:
219         dcd_sect3();
220     /*-------------------------------------------------------------------------*/
221     /*     DECODE SECT 4        : Product Definition Section                   */
222     /*-------------------------------------------------------------------------*/
223         printf("¥n<<SECTION 4>> : Product Definition Section¥n");
224     SECT4:
225         dcd_sect4();
226     /* Identify Data(elm lvl date period */
227         identifydata();
228     /*-------------------------------------------------------------------------*/
229     /*     DECODE SECT 5        : Data Representation Section                  */
230     /*-------------------------------------------------------------------------*/
231         printf("¥n<<SECTION 5>> : Data Representation Section¥n");
232         dcd_sect5();
233     /*-------------------------------------------------------------------------*/
234     /*     DECODE SECT 6        : Bit-map Section                              */
235     /*-------------------------------------------------------------------------*/
236         printf("¥n<<SECTION 6>> : Bit-map Section¥n");
237         dcd_sect6();
238     /*-------------------------------------------------------------------------*/
239     /*     DECODE SECT 7        : Data Section                                 */
240     /*-------------------------------------------------------------------------*/
241         printf("¥n<<SECTION 7>> : Data Section¥n");
242         dcd_sect7();
243         if( IFOUT       == 1 ) out_data();                  /*     OutPut Data   */
244     /*-------------------------------------------------------------------------*/
245     /*     DECODE SECT 8        : END or LOOP ?                                */
246     /*-------------------------------------------------------------------------*/
247         if( fread(buffer,sizeof(char), 4,fp) != 4 ){
248             printf("Read ERR !! File(%s)¥n",argv[1]);
249             exit(70);
250         }
251         if(buffer[0] == '7' && buffer[1] == '7' &&    buffer[2] == '7' && buffer[3] == '7'){
252             printf("GRIB END !!");
253             exit(0);
254             }
255         else{
256             wkl=convlong(&buffer[0], 4);
257             if( fread(buffer,sizeof(char), 1,fp) != 1 ){
258             printf("Read ERR !! File(%s)¥n",argv[1]);
259             exit(71);
260             }
```

```
261             fseek(fp,-5,1);                 /*   Back Space 5 bytes   */
262             ns=convlong(&buffer[0], 1);
263             switch(ns){
264                 case 2:
265                     printf("¥n----------¥n");
266                     printf("Record No=%d ¥n",++recno);
267                     printf("----------¥n");
268                     printf("<<SECTION 2>> : (Local Use Section)¥n");
269                     goto SECT2;
270                 case 3:
271                     printf("¥n----------¥n");
272                     printf("Record No=%d ¥n",++recno);
273                     printf("----------¥n");
274                     printf("¥n<<SECTION 3>> : Grid Definition Section¥n");
275                     goto SECT3;
276                 case 4:
277                     printf("¥n----------¥n");
278                     printf("Record No=%d ¥n",++recno);
279                     printf("----------¥n");
280                     printf("¥n<<SECTION 4>> : Product Definition Section¥n");
281                     goto SECT4;
282                 default:
283                     printf("ERROR in Section 8 !!");
284                     printf(" len ns = %d %d¥n",wkl,buffer[0]);
285                     exit(99);
286             }
287         }
288     }
289     /*=========================================================================*/
290     /*       END MAIN()       */
291     /*=========================================================================*/
292     /*    DEFINE FUNCTIONS                                                     */
293     /*=========================================================================*/
294     /*-------------------------------------------------------------------------*/
295     /*    DECODE SECTION 0                                                     */
296     /*-------------------------------------------------------------------------*/
297     void dcd_sect0(void)
298     {
299     /*    DECODE SECT 0                                                        */
300         /*   Check Header   */
301             if( fread(buffer,sizeof(char), 4, fp) == 0 ) exit(2);
302             while( buffer[0] != 'G' || buffer[1] != 'R' ||
303                    buffer[2] != 'I' || buffer[3] != 'B' ){
304                     printf("Cannot Find GRIB header !!¥n");
305
306                     fclose(fp);exit(99);
307                     }
308             printf("----------¥n");
309             printf("---- GRIB FILE !! -----¥n");
310             printf("----------¥n");
311         fread(buffer,sizeof(char), 12, fp);
312         mtn=convlong(&buffer[2], 1);
313         printf("¥n<<SECTION 0>> : Indicator Section¥n");
```

```
314         printf("GRIB Master Table Number : %d¥n",mtn);
315         printf("GRIB Edition Number          : %d¥n",buffer[3]);
316         printf("buffer[4]= %d¥n",convlong(&buffer[4],4));
317         if( convlong(&buffer[4],4)   == 0){
318               len = convlong(&buffer[8], 4);
319               printf("Total length of GRIB        : %d¥n",len);
320               }
321         else
322               {
323               printf("Larg Record !! Not Supported !!¥n¥n");
324               exit(3);
325               }
326   }
327
328   /*-------------------------------------------------------------------------*/
329   /*     DECODE SECTION 1                                                    */
330   /*-------------------------------------------------------------------------*/
331   void dcd_sect1(void)
332   {
333         unsigned long       ns1,idcenter,icsubc;
334
335         fread(buffer,sizeof(char), 5,fp);
336         len1=convlong(&buffer[0], 4);
337         ns1=convlong(&buffer[4], 1);
338         fread(buffer,sizeof(char),len1-5,fp);
339         idcenter=convlong(&buffer[0], 2);
340         icsubc=convlong(&buffer[2],2);
341         ltvn=convlong(&buffer[5],1);
342         iyyyy=convlong(&buffer[7], 2);
343         imm=convlong(&buffer[9], 1);
344         iday=convlong(&buffer[10], 1);
345         printf("Length of Section 1    : %d¥n",len1);
346         printf("Number of Section       : %d¥n",ns1);
347         printf("Identification of centre        : %d¥n",idcenter);
348         printf("Identification of sub-centre : %d¥n",icsubc);
349         printf("GRIB Master Tables Version Number : %d¥n",buffer[4]);
350         printf("GRIB Local Tables Version Number    : %d¥n",buffer[5]);
351         printf("Significance of Reference Time       : %d¥n",buffer[6]);
352         printf("Year                          : %d¥n",iyyyy);
353         printf("Month                          : %d¥n",imm);
354         printf("Day                            : %d¥n",iday);
355         printf("Hour                           : %d¥n",buffer[11]);
356         printf("Minute                         : %d¥n",buffer[12]);
357         printf("Second                         : %d¥n",buffer[13]);
358         printf("Production status of processed data : %d¥n",buffer[14]);
359         printf("Type of processed data                    : %d¥n",buffer[15]);
360
361         printf("-----------------------¥n");
362         printf("Record No=%d ¥n",++recno);
363         printf("-----------------------¥n");
364   }
365
366   /*-------------------------------------------------------------------------*/
```

```
367  /*     DECODE SECTION 2                                                    */
368  /*-------------------------------------------------------------------------*/
369  void dcd_sect2(void)
370  {
371       if( fread(buffer,sizeof(char), 5,fp) != 5 ){
372            printf("Read ERR SECT 2 !! File(%s)¥n",fname);
373            exit(72);
374            }
375       len2=convlong(&buffer[0], 4);
376       ns2=convlong(&buffer[4], 1);
377       printf("Length of Section 2    : %d¥n",len2);
378       printf("Number of Section      : %d¥n",ns2);
379       if( len2-4 != 0 ){
380            fread(buffer,sizeof(char),len2-5,fp);
381            printf(" Local use .... Not Supported!!");
382            exit(2);
383       }
384  }
385
386  /*-------------------------------------------------------------------------*/
387  /*     DECODE SECTION 3                                                    */
388  /*-------------------------------------------------------------------------*/
389  void dcd_sect3(void)
390  {
391       unsigned long wkl,i,ol,Ntoe;
392
393       if( fread(buffer,sizeof(char), 5,fp) != 5 ){
394            printf("Read ERR SECT 3 !! File(%s)¥n",fname);
395            exit(73);
396            }
397       len3=convlong(&buffer[0], 4);
398       ns3=convlong(&buffer[4], 1);
399
400       fread(buffer,sizeof(char),len3-5,fp);
401       ijmx = convlong(&buffer[1], 4);
402       Ngdt = convlong(&buffer[7], 2);
403       ol     = convlong(&buffer[5], 1);
404       printf("Length of section                    : %d¥n",len3);
405       printf("Number of section                    : %d¥n",ns3);
406       printf("Source of grid definition            : %d¥n",buffer[0]);
407       printf("Number of data points                : %d¥n",ijmx);
408       printf("optional list                        : %d¥n",ol);
409       printf("Interpretation of list               : %d¥n",buffer[6]);
410       printf("Grid Definition Template Number : %d¥n",Ngdt);
411       switch(Ngdt){
412            case      0:
413                 printf("((Grid Definition Template 3.0))¥n");
414                 printf("Shape of the earth                                    : %d¥n",buffer[9]);
415                 printf("Scale factor of radius of spherical earth             : %d¥n",buffer[10]);
416                 wkl = convlong(&buffer[11], 4);
417                 printf("Scaled value of radius of spherical earth             : %d¥n",wkl);
418                 printf("Scale factor of major axis of oblate spheroid earth   : %d¥n",buffer[15]);
419                 wkl =    convlong(&buffer[16], 4);
```

```
420                 printf("Scaled value of major axis of oblate spheroid earth   : %d¥n",wkl);
421                 printf("Scale factor of minor axis of oblate spheroid earth   : %d¥n",buffer[20]);
422                 wkl = convlong(&buffer[21], 4);
423                 printf("Scaled value of minor axis of oblate spheroid earth   : %d¥n",wkl);
424                 Ni    = convlong(&buffer[25], 4);
425                 Nj    = convlong(&buffer[29], 4);
426                 printf("number of points along a parallel                     : %d¥n",Ni);
427                 printf("number of points along a meridian                     : %d¥n",Nj);
428                 wkl = convlong(&buffer[33], 4);
429                 printf("Basic angle of the initial production domain          : %d¥n",wkl);
430                 wkl = convlong(&buffer[37], 4);
431                 printf("Subdivisions of basic angle                           : %d¥n",wkl);
432                 La1 = convlong(&buffer[41], 4);
433                 Lo1 = convlong(&buffer[45], 4);
434                 printf("latitude of first grid point(La1)                     : %d¥n",La1);
435                 printf("longitude of first grid point(Lo1)                    : %d¥n",Lo1);
436                 printf("Resolution and component flags                        : %d¥n",buffer[49]);
437                 La2 = convlatlon(&buffer[50]);
438                 Lo2 = convlatlon(&buffer[54]);
439                 printf("latitude of last grid point(La2)                      : %d¥n",La2);
440                 printf("longitude of last grid point(Lo2)                     : %d¥n",Lo2);
441                 Di    = convlong(&buffer[58], 4);
442                 Dj    = convlong(&buffer[62], 4);
443                 printf("i direction increment(Di)                             : %d¥n",Di);
444                 printf("j direction increment(Dj)                             : %d¥n",Dj);
445                 printf("Scanning mode                                         : %d¥n",buffer[66]);
446                 break;
447             default      :
448                 printf("This Grid is Not Supported !!");
449                 exit(10);
450         }
451     }
452
453     /*-------------------------------------------------------------------------*/
454     /*     DECODE SECTION 4                                                    */
455     /*-------------------------------------------------------------------------*/
456     void dcd_sect4(void)
457     {
458         unsigned long       noc,tcut,svl1,svl2;
459         unsigned long       iyyyye,imme,idaye,Nt,nmiss,tinc;
460         unsigned long       i,latn,lats,lone,lonw,Nc,vstd,vdist;
461                     char        sfl1;
462
463         if( fread(buffer,sizeof(char), 5,fp) != 5 ){
464             printf("Read ERR SECT 4 !! File(%s)¥n",fname);
465             exit(74);
466             }
467         len4=convlong(&buffer[0], 4);
468         ns4=convlong(&buffer[4], 1);
469
470         fread(buffer,sizeof(char),len4-5,fp);
471         noc = convlong(&buffer[0], 2);
472         Npdt= convlong(&buffer[2], 2);
```

```
473         printf("Length of section 4                                    : %d¥n",len4);
474         printf("Number of section                                       : %d¥n",ns4);
475         printf("Number of coordinates values after Template   : %d¥n",noc);
476         printf("Product Definition Template Number                 : %d¥n",Npdt);
477         printf("(( Product Definition Template 4.%d ))¥n",Npdt);
478         switch(Npdt){
479             case      2:                        /*    PDT4.2      */
480                 pc       = convlong(&buffer[4], 1);
481                 pn       = convlong(&buffer[5], 1);
482                 tcut     = convlong(&buffer[9], 2);
483                 kt       = convlong(&buffer[13], 4);
484                 tys      = convlong(&buffer[17], 1);
485                 sfl1     = buffer[18];
486                 svl1     = convlong(&buffer[19], 4);
487                 vl1      = svl1*pow(10.0,(double)sfl1);
488                 svl2     = convlong(&buffer[25], 4);
489                 dfn      = convlong(&buffer[29], 1);
490                 nmem     = -1;
491                 printf("Parameter category                                  : %d¥n",pc);
492                 printf("Parameter number                                    : %d¥n",pn);
493                 printf("Type of generating process                          : %d¥n",buffer[6]);
494                 printf("Background generating process identifier            : %d¥n",buffer[7]);
495                 printf("Forecast generating process identifier              : %d¥n",buffer[8]);
496                 printf("Hours after reference time of data cut-off          : %d¥n",tcut);
497                 printf("Minutes after reference time of data cut-off        : %d¥n",buffer[11]);
498                 printf("Indicator of unit of time range                     : %d¥n",buffer[12]);
499                 printf("Forecast time in units defined by octet 18          : %d¥n",kt);
500                 printf("Type of first fixed surface                         : %d¥n",tys);
501                 printf("Scale factor of first fixed surface                 : %d¥n",sfl1);
502                 printf("Scaled value of first fixed surface                 : %d¥n",svl1);
503                 printf("Type of second fixed surface                        : %d¥n",buffer[23]);
504                 printf("Scale factor of second fixed surface                : %d¥n",buffer[24]);
505                 printf("Scaled value of second fixed surface                : %d¥n",svl2);
506                 printf(" Derived forecast (see Code Table 4.7)              : %d¥n",dfn);
507                 printf("Number of forecasts in ensemble                     : %d¥n",buffer[30]);
508                 break;
509             case     11:                        /*    PDT4.11     */
510                 pc       = convlong(&buffer[4], 1);
511                 pn       = convlong(&buffer[5], 1);
512                 tcut     = convlong(&buffer[9], 2);
513                 kt       = convlong(&buffer[13], 4);
514                 tys      = convlong(&buffer[17], 1);
515                 sfl1     = buffer[18];
516                 svl1     = convlong(&buffer[19], 4);
517                 vl1      = svl1*pow(10.0,(double)sfl1);
518                 svl2     = convlong(&buffer[25], 4);
519                 tyens    = convlong(&buffer[29], 1);
520                 nmem     = convlong(&buffer[30], 1);
521                 iyyyye = convlong(&buffer[32], 2);
522                 imme     = convlong(&buffer[34], 1);
523                 idaye    = convlong(&buffer[35], 1);
524                 Nt       = convlong(&buffer[39], 1);
525                 nmiss    = convlong(&buffer[40], 4);
```

```
526             printf("Parameter category                                  : %d¥n",pc);
527             printf("Parameter number                                    : %d¥n",pn);
528             printf("Type of generating process                          : %d¥n",buffer[6]);
529             printf("Background generating process identifier            : %d¥n",buffer[7]);
530             printf("Forecast generating process identifier              : %d¥n",buffer[8]);
531             printf("Hours after reference time of data cut-off          : %d¥n",tcut);
532             printf("Minutes after reference time of data cut-off        : %d¥n",buffer[11]);
533             printf("Indicator of unit of time range                     : %d¥n",buffer[12]);
534             printf("Forecast time in units defined by octet 18          : %d¥n",kt);
535             printf("Type of first fixed surface                         : %d¥n",tys);
536             printf("Scale factor of first fixed surface                 : %d¥n",sfl1);
537             printf("Scaled value of first fixed surface                 : %d¥n",svl1);
538             printf("Type of second fixed surface                        : %d¥n",buffer[23]);
539             printf("Scale factor of second fixed surface                : %d¥n",buffer[24]);
540             printf("Scaled value of second fixed surface                : %d¥n",svl2);
541             printf("Type of ensemble forecast                           : %d¥n",buffer[29]);
542             printf("Perturbation number                                 : %d¥n",buffer[30]);
543             printf("Number of forecasts in ensemble                     : %d¥n",buffer[31]);
544             printf("Year                                : %d¥n",iyyyye);
545             printf("Month                               : %d¥n",imme);
546             printf("Day                                 : %d¥n",idaye);
547             printf("Hour                                : %d¥n",buffer[36]);
548             printf("Minut                               : %d¥n",buffer[37]);
549             printf("Second                              : %d¥n",buffer[38]);
550             printf("n - Number of time range            : %d¥n",Nt);
551             printf("Total number of data values missing                 : %d¥n",nmiss);
552             for(i=0 ; i < Nt ; ++i){
553                 lt      = convlong(&buffer[47+i*12], 4);
554                 tinc    = convlong(&buffer[52+i*12], 4);
555                 printf("Statistical process used to calculate ....      : %d¥n",buffer[44+i*12]);
556                 printf("Type of time increment                              : %d¥n",buffer[45+i*12]);
557                 printf("Indicator of unit of time for time range            : %d¥n",buffer[46+i*12]);
558                 printf("Length of the time range                            : %d¥n",lt);
559                 printf("Indicator of unit of time for the increment         : %d¥n",buffer[51+i*12]);
560                 printf("Time increment between successive fields            : %d¥n",tinc);
561             }
562             break;
563         case    12:                     /*    PDT4.12    */
564             pc      = convlong(&buffer[4], 1);
565             pn      = convlong(&buffer[5], 1);
566             tcut    = convlong(&buffer[9], 2);
567             kt      = convlong(&buffer[13], 4);
568             tys     = convlong(&buffer[17], 1);
569             sfl1    = buffer[18];
570             svl1    = convlong(&buffer[19], 4);
571             vl1     = svl1*pow(10.0,(double)sfl1);
572             svl2    = convlong(&buffer[25], 4);
573             dfn     = convlong(&buffer[29], 1);
574             nmem    = -1;
575             iyyyye = convlong(&buffer[31], 2);
576             imme    = convlong(&buffer[33], 1);
577             idaye   = convlong(&buffer[34], 1);
578             Nt      = convlong(&buffer[38], 1);
```

```
579                     nmiss   = convlong(&buffer[39], 4);
580                     printf("Parameter category                                  : %d¥n",pc);
581                     printf("Parameter number                                    : %d¥n",pn);
582                     printf("Type of generating process                          : %d¥n",buffer[6]);
583                     printf("Background generating process identifier            : %d¥n",buffer[7]);
584                     printf("Forecast generating process identifier              : %d¥n",buffer[8]);
585                     printf("Hours after reference time of data cut-off          : %d¥n",tcut);
586                     printf("Minutes after reference time of data cut-off        : %d¥n",buffer[11]);
587                     printf("Indicator of unit of time range                     : %d¥n",buffer[12]);
588                     printf("Forecast time in units defined by octet 18          : %d¥n",kt);
589                     printf("Type of first fixed surface                         : %d¥n",tys);
590                     printf("Scale factor of first fixed surface                 : %d¥n",sfl1);
591                     printf("Scaled value of first fixed surface                 : %d¥n",svl1);
592                     printf("Type of second fixed surface                        : %d¥n",buffer[23]);
593                     printf("Scale factor of second fixed surface                : %d¥n",buffer[24]);
594                     printf("Scaled value of second fixed surface                : %d¥n",svl2);
595                     printf(" Derived forecast (see Code Table 4.7)              : %d¥n",dfn);
596                     printf("Number of forecasts in ensemble                     : %d¥n",buffer[30]);
597                     printf("Year                                    : %d¥n",iyyyye);
598                     printf("Month                                   : %d¥n",imme);
599                     printf("Day                                     : %d¥n",idaye);
600                     printf("Hour                                    : %d¥n",buffer[35]);
601                     printf("Minut                                   : %d¥n",buffer[36]);
602                     printf("Second                                  : %d¥n",buffer[37]);
603                     printf("n - Number of time range        : %d¥n",Nt);
604                     printf("Total number of data values missing                 : %d¥n",nmiss);
605                     for(i=0 ; i < Nt ; ++i){
606                         lt      = convlong(&buffer[46+i*12], 4);
607                         tinc    = convlong(&buffer[51+i*12], 4);
608                         printf("Statistical process used to calculate ....      : %d¥n",buffer[43+i*12]);
609                         printf("Type of time increment                              : %d¥n",buffer[44+i*12]);
610                         printf("Indicator of unit of time for time range        : %d¥n",buffer[45+i*12]);
611                         printf("Length of the time range                            : %d¥n",lt);
612                         printf("Indicator of unit of time for the increment     : %d¥n",buffer[50+i*12]);
613                         printf("Time increment between successive fields        : %d¥n",tinc);
614                     }
615                     break;
616                 default:
617                     printf("This Product Definition Template is Not Supported !!");
618                     exit(11);
619             }
620     }
621     /*-------------------------------------------------------------------------*/
622     /*     DECODE SECTION 5                                                    */
623     /*-------------------------------------------------------------------------*/
624     void dcd_sect5(void)
625     {
626         fread(buffer,sizeof(char), 5,fp);
627         len5=convlong(&buffer[0], 4);
628         ns5=convlong(&buffer[4], 1);
629         fread(buffer,sizeof(char),len5-5,fp);
630         ijdim = convlong(&buffer[0], 4);
631         ntemplate = convlong(&buffer[4], 2);
```

```
632
633         printf("Length of section in octets                    : %d¥n",len5);
634         printf("Number of section                              : %d¥n",ns5);
635         printf("Number of data points                          : %d¥n",ijdim);
636         printf("Data Representation Template Number            : %d¥n",ntemplate);
637         switch(ntemplate){
638               case      0:
639                     R = convfloat(&buffer[6], 4);
640                     E = convlong(&buffer[10], 2);
641                           if(E > 32768) E = (E - 32768 )*(-1);
642                     D = convlong(&buffer[12], 2);
643                           if(D > 32768) D = (D - 32768 )*(-1);
644                     nbit = convlong(&buffer[14], 1);
645                     printf("Reference value (R)                          : %f¥n",R);
646                     printf("Binary scale factor (E)                      : %d¥n",E);
647                     printf("Decimal scale factor (D)                     : %d¥n",D);
648                     printf("Number of bits .....                         : %d¥n",nbit);
649                     printf("Type of original field values                : %d¥n",buffer[15]);
650                     break;
651               default:
652                           printf("This Data Representation Template is Not Supported!!");
653                           exit(12);
654               }
655   }
656
657   /*---------------------------------------------------------------------------*/
658   /*     DECODE SECTION 6                                                      */
659   /*---------------------------------------------------------------------------*/
660   void dcd_sect6(void)
661   {
662         int       i;
663
664         fread(buffer,sizeof(char), 5,fp);
665         len6=convlong(&buffer[0], 4);
666         ns6=convlong(&buffer[4], 1);
667         fread(buffer,sizeof(char), 1,fp);
668         ifbitmap = convlong(&buffer[0], 1);
669         printf("Length of section in octets                    : %d¥n",len6);
670         printf("Number of section                              : %d¥n",ns6);
671         printf("Bit-map indicator                              : %d¥n",ifbitmap);
672         for( i = 0 ; i < SIZEOFBITBUF ;++i){
673               map[i]=0xFF;
674               }
675         switch(ifbitmap){
676         case          0:
677               if ( len6-6 != fread(map,sizeof(char),len6-6,fp)){
678                     printf("Read ERR SECT 6-1 !! File(%s)¥n",fname);
679                     exit(100);
680                     }
681               break;
682               case      255:
683                     printf("No BITMAP !!¥n");
684                     break;
```

```
685             default:
686                 printf("This Bit-map indicator is Not Supportted!!");
687                 exit(13);
688         }
689     }
690
691     /*-------------------------------------------------------------------------*/
692     /*     DECODE SECTION 7                                                    */
693     /*-------------------------------------------------------------------------*/
694     void dcd_sect7(void)
695     {
696         unsigned char       msk_bit=MASK1;
697         unsigned long       posi,lgpv,i;
698
699
700         fread(buffer,sizeof(char), 5,fp);
701         len7=convlong(&buffer[0], 4);
702         ns7=convlong(&buffer[4], 1);
703         printf("Length of section in octets                    : %d¥n",len7);
704         printf("Number of section                                 : %d¥n",ns7);
705         fread(buffer2,sizeof(char),len7-5,fp);
706         /*printf("ntemplate=%d¥n",ntemplate);*/
707         switch(ntemplate){
708             case 0:
709
710                 posi = 0;
711                 for( i = 0 ;i < ijmx ;i++ ){
712                     nb = i / 8;
713                     mb = i % 8;
714                     if( 0 ==    (map[nb] & ( msk_bit >> mb ))){
715                         data[i]=UNDEF;
716                         }
717                     else{
718                         lgpv=longbybit(buffer2,&posi,nbit);
719                         data[i] = (R + lgpv*pow(2.0,(double)E))/pow(10.0,(double)D);
720                         }
721                     }
722
723                 printf("Check DATA !!       ¥n");
724                 printf("     No         value ¥n");
725                 for( i = 0; i < 10 ; ++i){
726                     printf("    %5d       %f¥n",i+1,data[i]);
727                 }
728                 printf("¥n¥n");
729                 for( i = ijmx-10; i < ijmx ; ++i){
730                     printf("    %5d       %f¥n",i+1,data[i]);
731                 }
732
733                 break;
734             default:
735                 printf("Unsupported Template !!");
736                 exit(14);
737         }
```

```
738  }
739  /*-------------------------------------------------------------------------*/
740  /*     OUT_DATA                                                            */
741  /*-------------------------------------------------------------------------*/
742  void out_data(void)
743  {
744       int                  itot;
745       unsigned int         i,j,ii,flen;
746       FILE                  *fpc1;
747       char                  flc1[30];
748       char                  celem[100],mmyyyy[7];
749       float                 fla1,fdi,flo1,fdj;
750
751       if(( fpg = fopen(fl,"wb")) == NULL ){
752            printf("Stop, Create File(%s)¥n",fl);
753            exit(99);
754            }
755
756       switch(Ngdt)
757            {
758            case 0:
759                 if( fwrite(data,4,ijmx,fpg) == ijmx )
760                 {
761                      tcount=1;
762                      }
763                 else{
764                      printf("Write ERR !! File(%s)¥n",fl);
765                      exit(98);
766                      }
767                 break;
768            default:
769                 printf(" This GDT is not supported !!    Ngdt= %d ¥n",Ngdt);
770                 exit(95);
771            }
772       if ( fclose(fpg) != 0 ){
773            printf("Close ERR !! File(%s)¥n",fl);
774            exit(97);
775            }
776  }
777
778  /*=========================================================================*/
779  /*      TOOLS                                                              */
780  /*=========================================================================*/
781
782  void usage(void)
783  {
784            puts("¥n");
785            puts("Usage : GRIB2 <FILE-NAME>");
786  }
787
788  long longbybit(unsigned char *buf,unsigned long *pos,unsigned long nbit)
789  {
790       char      i,ii;
```

```
791         long       pos8,mg8,rem8,nbytes;
792         long                   lout=0;
793         long                   lbig;
794         unsigned char       *buf2,*cbig;
795         unsigned char       mask,full=0xff;
796
797         lbig=1;
798         cbig = (unsigned char *)&lbig;
799
800         buf2 = (unsigned char *)&lout;
801         pos8 = *pos / 8;
802         mg8   = *pos % 8;
803
804         if( (*pos + nbit) % 8 == 0 ) rem8 = 0;
805         else                                  rem8 = 8 - (( *pos + nbit ) % 8 );
806
807         nbytes = (mg8 + nbit + rem8 ) / 8;
808         mask     = full >>    mg8 + rem8;
809
810         for( i = 0;i < nbytes -1 ;i++ ){
811              if( 1 == cbig[3] & 0x01 ) ii = 3 -i;
812              else                                  ii = i     ;
813              buf2[ii] = ( buf[pos8 + nbytes -1 - i] >> rem8
814                           | buf[pos8 + nbytes -1 - i - 1] << 8 - rem8 );
815         }
816         if( 1 == cbig[3] & 0x01 ) ii = 4 - nbytes;
817         else                                 ii = nbytes -1;
818         buf2[ii] = ( buf[pos8] >> rem8 & mask );
819
820         *pos = *pos + nbit;
821         return(lout);
822    }
823
824    long convlong( unsigned char *string, int nbyte )
825    {
826         int                  i,ii;
827         long                 numb=0;
828         long                 lbig;
829         unsigned char     *chrwrk,*cbig;
830
831         lbig=1;
832         cbig = (unsigned char *)&lbig;
833         /*if( 1 == cbig[3] & 0x01 ) printf("BIG ENDIAN !!");*/
834         chrwrk = (unsigned char *)&numb;
835         for( i = 0; i < nbyte; i++ ) {
836              if( 1 == cbig[3] & 0x01 ) ii = 4 - nbyte + i;
837              else                                  ii = nbyte - 1 - i;
838
839              chrwrk[ii] = string[i];
840         }
841         return( numb );
842    }
843
```

```
844  float convfloat( unsigned char *string, int nbyte )
845  {
846       int                  i,ii;
847       float                numb=0.;
848       long                 lbig;
849       unsigned char     *chrwrk,*cbig;
850
851       lbig=1;
852       cbig = (unsigned char *)&lbig;
853
854       chrwrk = (unsigned char *)&numb;
855       for( i = 0; i < nbyte; i++ ) {
856            if( 1 == cbig[3] & 0x01 ) ii = 4 - nbyte + i;
857            else                      ii = nbyte - 1 - i;
858
859            chrwrk[ii] = string[i];
860       }
861       return( numb );
862  }
863
864  long convlatlon(unsigned char *string )
865  {
866       long      ll;
867       char      c,a;
868       unsigned char wk[5];
869
870       c = MASK1 & string[0];
871       if( 0 != c ){
872            a = -1;}
873       else{
874            a = 1;
875       }
876
877       wk[0] = MASK7 & string[0];
878       wk[1]=string[1];wk[2]=string[2];wk[3]=string[3];
879       ll = (long)a*convlong(wk,(int)4);
880       return(ll);
881  }
882
883  /*=============================================================================
884  */
885  /*      used in identifydata                                                   */
886  /*=============================================================================
887  */
888  void fproduct(void)
889       {
890       char      wk[4];
891       fl[0]='X';
892       fl[1]='X';
893       fl[2]='X';
894       if( mtn == 0)
895         {
896            if( pc == 0 && pn == 0 ){fl[0]='T';fl[1]='_';fl[2]='_';}
```

```
897             if( pc == 0 && pn == 49){fl[0]='P';fl[1]='W';fl[2]='G';}
898             if( pc == 0 && pn == 9 ){fl[0]='T';fl[1]='A';fl[2]='_';}
899             if( pc == 1 && pn == 0 ){fl[0]='Q';fl[1]='Q';fl[2]='_';}
900             if( pc == 1 && pn == 1 ){fl[0]='R';fl[1]='H';fl[2]='_';}
901             if( pc == 1 && pn == 8 ){fl[0]='R';fl[1]='R';fl[2]='R';}
902             if( pc == 1 && pn ==20 ){fl[0]='r';fl[1]='r';fl[2]='r';}
903             if( pc == 1 && pn ==210){fl[0]='R';fl[1]='R';fl[2]='d';}
904             if( pc == 1 && pn ==211){fl[0]='R';fl[1]='A';fl[2]='d';}
905             if( pc == 1 && pn ==212){fl[0]='Q';fl[1]='Q';fl[2]='A';}
906             if( pc == 1 && pn ==213){fl[0]='R';fl[1]='H';fl[2]='A';}
907             if( pc == 2 && pn == 1 ){fl[0]='W';fl[1]='_';fl[2]='_';}
908             if( pc == 2 && pn == 2 ){fl[0]='U';fl[1]='_';fl[2]='_';}
909             if( pc == 2 && pn == 3 ){fl[0]='V';fl[1]='_';fl[2]='_';}
910             if( pc == 2 && pn == 4 ){fl[0]='P';fl[1]='S';fl[2]='I';}
911             if( pc == 2 && pn == 5 ){fl[0]='C';fl[1]='H';fl[2]='I';}
912             if( pc == 2 && pn ==210){fl[0]='U';fl[1]='A';fl[2]='_';}
913             if( pc == 2 && pn ==211){fl[0]='V';fl[1]='A';fl[2]='_';}
914             if( pc == 3 && pn == 0 ){fl[0]='P';fl[1]='_';fl[2]='_';}
915             if( pc == 3 && pn == 1 ){fl[0]='P';fl[1]='s';fl[2]='a';}
916             if( pc == 3 && pn == 4 ){fl[0]='z';fl[1]='_';fl[2]='_';}
917             if( pc == 3 && pn == 5 ){fl[0]='Z';fl[1]='_';fl[2]='_';}
918             if( pc == 3 && pn == 8 ){fl[0]='P';fl[1]='A';fl[2]='_';}
919             if( pc == 3 && pn == 9 ){fl[0]='Z';fl[1]='A';fl[2]='_';}
920             }
921         else if(mtn == 10 )
922             {
923             if( pc == 3 && pn == 0 ){fl[0]='S';fl[1]='T';fl[2]='_';}
924             if( pc == 3 && pn ==192){fl[0]='S';fl[1]='T';fl[2]='A';}
925             }
926         fl[3]='?';
927         fl[4]='?';
928         fl[5]='?';
929         if( nmem <     0 ){fl[3]='_';fl[4]='_';fl[5]='_';}
930         if( nmem >=    0 && nmem < 100 ){
931             switch (tyens)
932                 {
933                 case       1:
934                   fl[5] = '_'; break;
935                 case       2:
936                   fl[5] = 'm'; break;
937                 case       3:
938                   fl[5] = 'p'; break;
939                 default:
940                   printf(" tyens Not Supported !! tyens=%d ¥n",tyens);exit(71);
941                 }
942             sprintf(wk,"%02d",nmem);fl[3]=wk[0];fl[4]=wk[1];
943             }
944         if( nmem >= 100 ){printf(" Over 100 members!! Not Supported.¥n");exit(70);}
945
946         }
947
948     void flevel(void)
949         {
```

```
950         char      wk[10];
951         unsigned long vl1hpa;
952         vl1hpa=vl1*0.01;
953         fl[6]='X';
954         fl[7]='X';
955         fl[8]='X';
956         if( tys == 1 ){fl[6]='S';fl[7]='F';fl[8]='C';}
957         if( tys == 100 ){sprintf(wk,"%03d",vl1hpa);fl[6]=wk[0];fl[7]=wk[1];fl[8]=wk[2];}
958         if( tys == 101 ){fl[6]='S';fl[7]='E';fl[8]='A';}
959         if( tys == 103 ){fl[6]='2';fl[7]='M';fl[8]='_';}
960
961         }
962     void fderived(void)
963         {
964         fl[9]='X';fl[10]='X';fl[11]='X';
965         if( dfn == 0 ){fl[9]='E';fl[10]='S';fl[11]='_';}/* Unweighted mean of all members              */
966         if( dfn == 1 ){fl[9]='E';fl[10]='S';fl[11]='W';}/* Weighted mean of all members               */
967         if( dfn == 4 ){fl[9]='S';fl[10]='P';fl[11]='R';}/* Spread of all members                      */
968         if( dfn == 5 ){fl[9]='L';fl[10]='A';fl[11]='I';}/* Large Anomaly Index of all members         */
969         if( dfn == 6 ){fl[9]='C';fl[10]='L';fl[11]='S';}/* Unweighted mean of the cluster members */
970
971         }
972     void fhemispher(void)
973         {
974         fl[12]='?';fl[13]='?';
975         if(La1 >= 80000000 && La2 <= -80000000 ){fl[12]='-';fl[13]='G';}
976         if(La1 == 90000000 && La2 == 0 ) {fl[12]='-';fl[13]='N';}
977         if(La1 == 0              && La2 == -90000000 ){fl[12]='-';fl[13]='S';}
978         if(La1 == 90000000 && La2 == -90000000 ){fl[12]='-';fl[13]='G';}
979
980         }
981     void fperiod(void)
982         {
983         fl[14]='.';fl[15]='F';fl[16]='?';fl[17]='?';fl[18]='E';fl[19]='?';
984         fl[20]='?';fl[21]='?';fl[22]='?';fl[23]='?';fl[24]='?';
985         sprintf(&fl[14],".F%03dE%03d",kt,lt);
986
987         }
988
989     /*=============================================================================
990     */
991     /*     Identify Data (FILE NAME)                                              */
992     /*=============================================================================
993     */
994     void identifydata(void)
995         {
996         int       i;
997         for( i = 0 ; i < 30 ; ++i){
998               fl[i]=' ';
999               }
1000        fproduct();
1001        flevel();
1002        fderived();
```

```
1003        fhemispher();
1004        fperiod();
1005        fl[25]=0x00;
1006        }
1007
1008
```