File under: A Date: 22 Jan 1985 Name: John Strawn

Name: John Strawn

Project: J Programmer: AWN

File Name: UDP2:EMERGE.SAI[LIB,AWN]

File Last Written: 20:09 21 Jan 1985

Time: 12:43 Date: 22 Jan 1985

Center for Computer Research
in Music and Acoustics
Department of Music
Stanford, California

This source code in the "SAIL" language was written using the "E" text editor
available at the time at the Stanford Artificial Intelligence Laboratory (SAIL).
This page is the table of contents, see the "Page" column.


```
COMMENT ⊗   VALID 00011 PAGES
C REC  PAGE   DESCRIPTION
C00001 00001
C00002 00002    IFCR NOT DECLARATION(garply) THENC
C00005 00003    !! which parts of emerge shall we load?
C00017 00004    !! then load those features
C00023 00005    !! declarations, version number
C00028 00006    !! mrgExtensions
C00030 00007    !! assemble main menu, no matter what the version
C00035 00008    !! the buck starts here:  initialization
C00039 00009    !! main loop
C00045 00010    !! breadBoard version of EMERGE
C00048 00011    END   "emerge"
C00049 ENDMK
C⊗;
```

```
IFCR NOT DECLARATION(garply) THENC
BEGIN "emerge"
COMMENT this IFCR...ENDC block, embedded in a separate file, allows
for conditional compilation with SAIL;

REQUIRE "dsk:sail.awn[1,awn]" SOURCE!FILE;
DEFINE NO_ITEMVARS_PLEASE = TRUE;
DEFINE Dont_Require_AWNLIB_Dammit = TRUE;
DEFINE Dont_Require_FILEIO_Dammit = TRUE;
DEFINE Dont_Require_Grnlib_Dammit = TRUE;
DEFINE Dont_Require_Menu_Dammit = TRUE;
DEFINE Dont_Require_Merge_Dammit = TRUE;
REQUIRE "dsk:awnlib.hdr[sub,sys]" SOURCE!FILE;
REQUIRE "dsk:merge.hdr[lib,awn]" SOURCE!FILE;
REQUIRE "dsk:emerge.hdr[lib,awn]" SOURCE!FILE;
REQUIRE "dsk:menu.hdr[SUB,SYS]" SOURCE!FILE;
REQUIRE myDev&"hmerge.REL[lib,awn]" LOAD_MODULE;
REQUIRE myDev&"hMENU.REL[lib,awn]" LOAD_MODULE;
REQUIRE myDev&"hileio.rel[lib,awn]" LIBRARY;
REQUIRE myDev&"hpprox.rel[lib,awn]" LIBRARY;
REQUIRE myDev&"hgrnlb.rel[sub,sys]" LIBRARY;
REQUIRE myDev&"hwnlib.rel[lib,awn]" LIBRARY;
REQUIRE crlf&"hgrnlb from pit:" MESSAGE;

DEFINE GARPLY=TRUE;
ENDC

EXTERNAL INTEGER dont_move;
COMMENT this is a bug trap from LEDWIN in GRNLIB.  If the damn LOADER
ever complains about can't find it, then just delete this
declaration and the assignment dont_move←TRUE made somewhere
in this file.  Without this bug trap, it can sometimes happen
that the window will move around in funny ways, especially when
you're editing a line.  Added 10/29/83;

REQUIRE 500 STRING_PDL;
```

```
!! which parts of emerge shall we load?;

DEFINE breadboard = FALSE;      !! normally FALSE;
IFC breadBoard THENC REQUIRE crlf&"********** breadboard on ********** " MESSAGE; ENDC
!! set breadBoard to TRUE to avoid the main menu.  Emerge then just steps
through whatever option(s) you're loading, one at a time.  breadBoard =
FALSE is the usual setting.  breadBoard = TRUE intended for debugging
only. For Breadboard=TRUE, it is intended that emergeVersion (below) be
TRUE;

COMMENT this set of compile-time switches serve as master switches
for the various incarnations of EMERGE.  Only one of these should be TRUE;
DEFINE
   systemEmergeVersion = TRUE,  COMMENT for SYS:EMERGE---normal setting;
   emergeVersion  = FALSE,      COMMENT for UDP2:EMERGE.DMP[lib,awn], for debugging;
   mfEDVersion    = FALSE,      COMMENT for SYS:MFED;
   mf2EDVersion   = FALSE,      COMMENT for editing two funcs at once;
   mfApprVersion = FALSE;       COMMENT for SYS:MFAPPR;

COMMENT the following sets of compile-time switches, all of which begin
with "do...", allow you to turn on or off loading the various emerge
options.  In particular, these compile-time switches control 1) whether
the offending load module is REQUIREd --- see next page 2) whether the
option is mentioned at all in the main menu --- see the main menu for how
that works.  If you want to tweek these by hand, then set emergeVersion
(above) to TRUE, and change whichever do.... to FALSE.;

IFC emergeVersion THENC         COMMENT this is the complete list of all that's
                                        available;
REQUIRE crlf&"setting up as emerge " MESSAGE;
DEFINE
  doShow        = FALSE,        COMMENT Show 1 channel;
  doThreeD      = TRUE ,        COMMENT 3-d plot;
  doThreeD2     = FALSE,        COMMENT 3-d plot for 2 funcs;
  doSpect       = FALSE,        COMMENT Spectrographic plot;
  doSplMer      = FALSE,        COMMENT Pavlidis' Split/Merge Approximations;
  doEdSeg       = FALSE,        COMMENT Emerge Function editor;
  doedSeg2      = FALSE,        COMMENT Emerge Function editor for 2 funcs;

  doSlice       = FALSE,        COMMENT slice of life;
  doEfl         = FALSE,        COMMENT Get current input merge file;
  doDir         = FALSE,        COMMENT Print directory of current input merge file;
  doFDef        = FALSE,        COMMENT Edit defaults for current merge file;
  doInFun       = TRUE ,        COMMENT Read in .FUN file;
  doOutFun      = FALSE,        COMMENT Write out .FUN file;
  doMfDif       = FALSE,        COMMENT dif between .MF and .FUN file;
  doNews        = FALSE;        COMMENT News;
ENDC

IFC systememergeVersion THENC
REQUIRE crlf&"setting up as system emerge " MESSAGE;
DEFINE
  doSlice       = TRUE ,        COMMENT slice of life;
  doEfl         = TRUE ,        COMMENT TRUE = Get current input merge file with menu;
                                COMMENT FALSE = user prompted for one .MF file at startup;
  doDir         = TRUE ,        COMMENT Print directory of current input merge file;
  doFDef        = TRUE ,        COMMENT Edit defaults for current merge file;
  doShow        = TRUE ,        COMMENT Show 1 channel;
  doThreeD      = TRUE ,        COMMENT 3-d plot;
  doThreeD2     = FALSE,        COMMENT 3-d plot for 2 funcs;
  doSpect       = TRUE ,        COMMENT Spectrographic plot;
  doSplMer      = TRUE ,        COMMENT Pavlidis' Split/Merge Approximations;
  doEdSeg       = TRUE ,        COMMENT Emerge Function editor;
  doedSeg2      = FALSE,        COMMENT Emerge Function editor for 2 funcs;
  doInFun       = TRUE ,        COMMENT Read in .FUN file;
  doOutFun      = TRUE ,        COMMENT Write out .FUN file;
  doMfDif       = FALSE,        COMMENT dif between .MF and .FUN file;
  doNews        = TRUE ;        COMMENT News;
ENDC
```

```
IFC mfEdVersion THENC
  REQUIRE crlf&"setting up as MFED " MESSAGE;
DEFINE
  doSlice       = FALSE,        COMMENT slice of life;
  doEfl         = FALSE,        COMMENT TRUE = Get current input merge file with menu;
                                COMMENT FALSE = user prompted for one .MF file at startup;
  doDir         = FALSE,        COMMENT Print directory of current input merge file;
  doFDef        = FALSE,        COMMENT Edit defaults for current merge file;
  doShow        = FALSE,        COMMENT Show 1 channel;
  doThreeD      = FALSE,        COMMENT 3-d plot;
  doThreeD2     = FALSE,        COMMENT 3-d plot for 2 funcs;
  doSpect       = FALSE,        COMMENT Spectrographic plot;
  doSplMer      = FALSE,        COMMENT Pavlidis' Split/Merge Approximations;
  doEdSeg       = TRUE ,        COMMENT Emerge Function editor;
  doedSeg2      = FALSE,        COMMENT Emerge Function editor for 2 funcs;
  doInFun       = TRUE ,        COMMENT Read in .FUN file;
  doOutFun      = TRUE ,        COMMENT Write out .FUN file;
  doMfDif       = FALSE,        COMMENT dif between .MF and .FUN file;
  doNews        = TRUE ;        COMMENT News;
ENDC

IFC mf2EdVersion THENC
  REQUIRE crlf&"setting up as MF2ED " MESSAGE;
  EXTERNAL PROCEDURE make2EdSeg(RECORD!POINTER(emergeClass) emergeRp1, emergeRp2);
DEFINE
  doMfDif       = FALSE,        COMMENT dif between .MF and .FUN file;
  doSlice       = FALSE,        COMMENT slice of life;
  doEfl         = FALSE,        COMMENT TRUE = Get current input merge file with menu;
                                COMMENT FALSE = user prompted for one .MF file at startup;
  doDir         = FALSE,        COMMENT Print directory of current input merge file;
  doFDef        = FALSE,        COMMENT Edit defaults for current merge file;
  doShow        = FALSE,        COMMENT Show 1 channel;
  doThreeD      = FALSE,        COMMENT 3-d plot;
  doThreeD2     = FALSE,        COMMENT 3-d plot for 2 funcs;
  doSpect       = FALSE,        COMMENT Spectrographic plot;
  doSplMer      = FALSE,        COMMENT Pavlidis' Split/Merge Approximations;
  doEdSeg       = FALSE,        COMMENT Emerge Function editor;
  doedSeg2      = TRUE ,        COMMENT Emerge Function editor for 2 funcs;
  doInFun       = TRUE ,        COMMENT Read in .FUN file;
  doOutFun      = TRUE ,        COMMENT Write out .FUN file;
  doNews        = TRUE ;        COMMENT News;
ENDC

IFC mfApprVersion THENC
  REQUIRE crlf&"setting up as MFAPPR " MESSAGE;
DEFINE
  doSlice       = FALSE,        COMMENT slice of life;
  doEfl         = FALSE,        COMMENT TRUE = Get current input merge file with menu;
                                COMMENT FALSE = user prompted for one .MF file at startup;
  doDir         = FALSE,        COMMENT Print directory of current input merge file;
  doFDef        = FALSE,        COMMENT Edit defaults for current merge file;
  doShow        = FALSE,        COMMENT Show 1 channel;
  doThreeD      = FALSE,        COMMENT 3-d plot;
  doThreeD2     = FALSE,        COMMENT 3-d plot for 2 funcs;
  doSpect       = FALSE,        COMMENT Spectrographic plot;
  doSplMer      = TRUE ,        COMMENT Pavlidis' Split/Merge Approximations;
  doEdSeg       = FALSE,        COMMENT Emerge Function editor;
  doedSeg2      = FALSE,        COMMENT Emerge Function editor for 2 funcs;
  doInFun       = TRUE ,        COMMENT Read in .FUN file;
  doOutFun      = TRUE ,        COMMENT Write out .FUN file;
  doMfDif       = FALSE,        COMMENT dif between .MF and .FUN file;
  doNews        = FALSE;        COMMENT News;
ENDC
```

```
!! then load those features;

COMMENT the rest of this page works "automatically."  Depending on the
settings of the "do..." compile-time switches on the previous page, appropriate
load modules are REQUIREd here.  The following are loaded in by .HDR files
from the first page:   MERGE.REL[lib,awn], APPROX.REL[LIB,AWN]
(alternatively:  HILEIO.REL[lib,awn], HPPROX.REL[lib,awn] for high-segment version)
;

REQUIRE myDev&"umerge.rel" LOAD!MODULE; COMMENT utility and miscellaneous routines;
IFC doFDef THENC REQUIRE myDev&"filede.rel[lib,awn]" LOAD!MODULE; ENDC
                COMMENT edits EMERGE default settings for current input merge file;
IFC doNews THENC REQUIRE myDev&"news.rel[lib,awn]" LOAD!MODULE; ENDC
                COMMENT shows EMERGE news;
IFC doSplMer THENC REQUIRE myDev&"appr.rel[lib,awn]" LOAD!MODULE;
                COMMENT the interface to Pavlidis line segment approximation
                        software.  See appr.sai for where the Pavlidis code lives.;
            ENDC
IFC doEFL
    THENC IFC NOT breadBoard THENC  REQUIRE myDev&"efl.rel[lib,awn]" LOAD!MODULE; ENDC
                COMMENT the file list editor;
    ELSEC IFC NOT breadBoard THENC REQUIRE myDev&"getif.rel[lib,awn]" LOAD!MODULE; ENDC
ENDC
IFC doDir    THENC REQUIRE myDev&"makdir.rel[lib,awn]" LOAD!MODULE; ENDC
                COMMENT shows the directory of the current input merge file;
IFC doslice    THENC REQUIRE myDev&"slice.rel[lib,awn]" LOAD!MODULE; ENDC
                COMMENT spectral slice of current input merge file;
IFC doSpect  THENC REQUIRE myDev&"spect.rel[lib,awn]" LOAD!MODULE; ENDC
                COMMENT makes a spectrographic plot;
IFC doShow   THENC REQUIRE myDev&"showon.rel[lib,awn]" LOAD!MODULE; ENDC
                COMMENT shows one channel of the current input merge file;
IFC doEdSeg  THENC
   REQUIRE mydev&"edseg.rel[lib,awn]" LOAD!MODULE;
                COMMENT the EMERGE interface to worked.sai;
   REQUIRE mydev&"cured.rel[lib,awn]" LOAD!MODULE;
                COMMENT the EMERGE function editor;
   ENDC
IFC doedSeg2 THENC
   REQUIRE mydev&"edseg2.rel[lib,awn]" LOAD!MODULE;
                COMMENT the EMERGE interface to cured2.sai;
   REQUIRE mydev&"cured2.rel[lib,awn]" LOAD!MODULE;
                COMMENT the EMERGE function editor for 2 functions;
   ENDC
IFC doInFun  THENC REQUIRE myDev&"getfun.rel[lib,awn]" LOAD!MODULE; ENDC
                COMMENT the EMERGE interface to segsyn;
IFC dothreeD THENC REQUIRE myDev&"make3d.rel[lib,awn]" LOAD!MODULE; ENDC
                COMMENT creates 3D plots;
IFC doSplMer OR  doOutFun OR doEdSeg OR doedSeg2 THENC REQUIRE myDev&"outfun.rel[lib,awn]" LOAD!MODULE
;  ENDC
                COMMENT the EMERGE interface to seg.sai;
IFC doMfDif THENC REQUIRE myDev&"mfdif.rel[lib,awn]" LOAD!MODULE; ENDC

IFC doThreeD OR doInFun OR doOutFun OR doShow OR doSplMer OR doEdSeg OR doedSeg2
        OR doThreeD2
   THENC REQUIRE myDev&"seg.rel[lib,awn]" LOAD!MODULE; ENDC
                COMMENT writes out .FUN files;

IFC
   doEdSeg  OR
   doedSeg2  THENC
REQUIRE myDev&"edpt.rel[lib,awn]" LOAD!MODULE; ENDC

IFC
   doSplMer OR
   doShow OR
   doEdSeg OR
   doedSeg2 OR
   doSpect OR
   doThreeD2 OR
```

```
   doThreeD THENC
REQUIRE myDev&"win.rel[lib,awn]" LOAD!MODULE;
ENDC
COMMENT the code which displays merge and .FUN functions;

IFC doThreed2
THEN REQUIRE crlf&"spurious mak2threed declaration " MESSAGE;
EXTERNAL PROCEDURE mak2ThreeD(RECORD!POINTER(emergeClass) emergeRp1, emergeRp2);
rEQUIRE myDev&"mak23d.rel[lib,awn]" LOAD!MODULE;
ENDC
```

```
!! declarations, version number;

INTERNAL BOOLEAN inDefault;  !! also examined by UMERGE utilities;
IFC doEFL THENC STRING defaultFileName; ENDC
                              !! file name for reading in, writing out merge file names;

IFC NOT breadBoard THENC

DEFINE
   emergeVersionNo = <("13.2")>;  COMMENT change this in NEWS.SAI too;
   REQUIRE crlf&"Version "&emergeVersionNo&crlf MESSAGE;

COMMENT The following are needed by the menu package for making the main menu;
DEFINE
   cmdCol = 10*pixelsPerChar,    COMMENT left-hand column to start commands in menu;
   menuTopLine = -1,             COMMENT place the main menu into
                                    default place on screen;
   help = <("Input: eMerge currently accepts .MF files for input."&crlf&
       crlf&
       "For the other options, you will be given another menu to play with")>;

IFC mfEdVersion
THENC DEFINE
   myTmpString = <("Welcome to MFED, a subset of eMerge (Version "&
                                          emergeVersionNo&")"&crlf&crlf)>;
ELSEC   IFC mf2EdVersion
        THENC DEFINE
           myTmpString = <("Welcome to MF2ED, a subset of eMerge (Version "&
                                          emergeVersionNo&")"&crlf&crlf)>;
        ELSEC IFC mfApprVersion
              THENC DEFINE
              myTmpString = <("Welcome to MFAppr, a subset of eMerge (Version "&
                                          emergeVersionNo&")"&crlf&crlf)>;
              ELSEC   !! system version, or my version;
                 DEFINE mymyTmpString=
                    <("Welcome to eMerge, the Merge File Editor (Version "&
                                             emergeVersionNo&")")>;
                     IFC systemEmergeVersion
                        !! THENC DEFINE myTmpString = <(mymyTmpString&
                        crlf&"Other parts of eMerge are in MFAPPR, MFED, and MFED2")>;
                     THENC DEFINE myTmpString = <mymyTmpString>;
                     ELSEC DEFINE myTmpString = <mymyTmpString>;
                     ENDC
              ENDC
        ENDC
ENDC

INTEGER lineNo,                    !! line number in main menu of option to be done;
   actChar;                        !! how did we return from pickOne?;

RECORD!POINTER(emergeClass) emergeInHead,
                                   !! points to linked list of input merge files;
   inEmergeRp;                     !! current input merge file;
                                   !! emergeClass defined in emerge.hdr;

IFC doEdseg2 OR doThreed2
THENC RECORD!POINTER (emergeClass) inEmergeRp2;
ENDC

IFC doefl
THENC
   DEFINE labelString = <myTmpString&crlf&crlf>;
ELSEC

    IFC doThreeD2 OR doEdSeg2
    THENC
        DEFINE labelString = <myTmpString&crlf&"Input file: "&
        mrgMakeFileName(emergeClass:sonRp[inEmergeRp])&crlf&
        (IF inEmergeRp2 = NULL!RECORD THEN "" ELSE
```

```
        "and "&mrgMakeFileName(emergeClass:sonRp[inEmergeRp2])&crlf)&
        crlf>;
    ELSEC
        DEFINE labelString = <myTmpString&crlf&"Input file: "&
        mrgMakeFileName(emergeClass:sonRp[inEmergeRp])&crlf&crlf>; ENDC
    ENDC

ENDC
```

```
!! mrgExtensions;

DEFINE includeMF = TRUE ;
DEFINE includeK  = FALSE;
DEFINE includePV = FALSE;       !! little code here;
DEFINE includeP  = FALSE;       !! no code here;
DEFINE includeZW = FALSE;       !! no code here;
!! one of the above MUST be true! ;
DEFINE includeOutMerge = TRUE ;
REQUIRE crlf&"This version set up for " MESSAGE;
IFC includePV THENC REQUIRE "PV " MESSAGE; ENDC
IFC includeMF THENC REQUIRE "MF " MESSAGE; ENDC
IFC includeK THENC REQUIRE "K " MESSAGE; ENDC
IFC includeZW THENC REQUIRE "ZW " MESSAGE; ENDC
REQUIRE crlf MESSAGE;

PRELOAD!WITH
IFC includeMf THENC     "MF", "MY", "AI", "AS", ENDC
IFC includePV THENC     "PV", ENDC
IFC includeK THENC      "K", ENDC
IFC includeZW THENC     "ZW", ENDC
        "MRG",
        "";
INTERNAL   STRING ARRAY mrgExtensions[1:
        2
        IFC includeMF THENC +4 ENDC
        IFC includePV THENC +1 ENDC
        IFC includeK THENC +1 ENDC
        IFC includeZw THENC +1 ENDC
        ];
```

```
!! assemble main menu, no matter what the version;

IFC NOT BREADBOARD THENC  !! whole page;

!! these little macros allow the menu array to be just big enough.;
REDEFINE counter = 1;   !! DEFINEd in SAIL.AWN;
DEFINE
   foo(bool) = <IFC bool AND needComma tHENC , ENDC IFC bool THENC>,
   bar = <REDEFINE needComma = TRUE; nextCounter; ENDC>;
COMMENT If "bool" is TRUE, then foo drops a comma in place (if needed)
        and goes ahead and inserts the line.  Bar tells us we'll need
        a comma in the next line, and finishes the IFC...THENC started by foo.
        It "of course" keeps track of the numbers of lines as it crunches along.
        To get a blank line, say foo(TRUE) "" bar;
DEFINE needComma = FALSE; !! no comma is needed after the last entry in the preload;

IFC doEfl THENC
DEFINE inputLine = <("   Get current input merge file "&"")>;
   !! changed in main menu during execution to include current input file name;
ENDC

!! assemble the main menu;
PRELOAD!WITH
   foo((doEfl OR doDir OR doFDef))
                "Dealing with Merge files:"                          bar
   foo(doEfl)   inputLine                                            bar
   foo(doDir)   "   Print directory of current input merge file"     bar
   foo(doFDef)  "   Edit defaults for current merge file"           bar
   foo((doEfl OR doDir OR doFDef)) ""                                bar
   foo((doShow OR doThreeD OR doSpect OR doSlice OR doThreed2))
                "Looking at data in current input .MF file:"        bar
   foo(doShow)  "   Show 1 channel"                                  bar
   foo(doThreeD) "  3-d plot"                                        bar
   foo(doThreeD2) "   3-d plot of 2 merge files "                   bar
   foo(doSpect)  "   Spectrographic plot"                           bar
   foo(doSlice)  "   Create spectral average "                      bar
   foo((doShow OR doThreeD OR doSpect OR doSlice OR doThreeD2)) "" bar
   foo((doSplMer OR doEdSeg OR doedSeg2 OR doMfDif))
                "Operating on data in current input .MF file:"      bar
   foo(doSplMer) "   Pavlidis' Split/Merge Approximations"          bar
   foo(doEdSeg)  "   Emerge Function editor"                        bar
   foo(doedSeg2)  "   Emerge Function editor, 2 functions"          bar
   foo(doMfDif)   "   Create difference between .MF and .FUN"      bar
   foo((doSplMer OR doEdSeg OR doedSeg2 OR doMfDif)) ""             bar
   foo((doInFun OR doOutFun))
                "Dealing with .FUN files:"                          bar
   foo(doInFun)  "   Read in .FUN file"                             bar
   foo(doOutFun) "   Write out .FUN file"                           bar
   foo((doInFun OR doOutFun)) ""                                    bar
   foo((doNews OR doDebug))
                "Emerge miscellany:"                                bar
   foo(doNews)   "   News"                                          bar
   foo(doDebug)  "   Debugging:  show list of input files";         bar;
SAFE STRING ARRAY mainMenu[1:counter-1];

ENDC
```

```
!! the buck starts here:  initialization;

IFC NOT BREADBOARD THENC  !! whole page;

dont_move ← TRUE;  !! see comment at beginning of file regarding this grnlib bug trap;

IFC FALSE THENC
IF getWizard    !! tell us if a wizard is using this thing;
   THEN inDefault ← TRUE    !! an initial setting, modified by procInputSwitches;
   ELSE inDefault ← FALSE;
ELSEC inDefault ← FALSE; ENDC
procInputSwitches;  !! in particular:  shall we load in the default input file?;
IFC doefl
THENC
   defaultFileName ← "MRG.LST/q";
ENDC

OUTSTR(crlf&"EMERGE Version "&emergeVersionNo&crlf);
lineno ← 2;  !! the first line in main menu is a labelling line;
IFC doEfl THENC mainMenu[2] ← inputLine; ENDC
emergeInHead ← inEmergeRp ← NULL!RECORD; !! this is here in case you restart
        by typing .S to the monitor;

!! if requested, try to read in input file;
IFC doEfl THENC
IF inDefault
THEN BEGIN
   STRING tmpStr;
   tmpStr ← "UDP2:c074.MY4[GRO,AWN]";
   OUTSTR("Trying to read input file "&tmpStr&" ");
   inEmergeRp ← NEW!RECORD(emergeClass);
   REQUIRE crlf&"sonRp for default input file name NEW!RECORDed to mfClass " MESSAGE;
   emergeClass:sonRp[inEmergeRp] ← NEW!RECORD(mfClass);
   mrgClass:readAlter[emergeClass:sonRp[inEmergeRp]] ← FALSE;
   IF (getInMergeFile(tmpStr,mfType,emergeClass:sonRp[inEmergerp],mrgExtensions,TRUE))
        !! verbose;
      = noIoFailure
   THEN BEGIN
      emergeInHead←inEmergeRp;
      IFC doEfl THENC
         mainMenu[2] ← inputLine & "("&
            (mrgMakeFileName(emergeClass:sonRp[inEmergeRp]))&")";
         ENDC
      END
   ELSE
      BEGIN !! can't find the default input file, ignore request gracefully;
      OUTSTR(crlf&"Can't find default input file ");
      deleteRecord(emergeClass:sonRp[inEmergeRp]);
      deleteRecord(inEmergeRp);
      emergeInHead ← inEmergeRp ← NULL!RECORD;
      END;
   END;

ELSEC

   inEmergeRp ← NEW!RECORD(emergeClass);
   emergeClass:sonRp[inEmergeRp] ← getIf;

   IFC doEdseg2 OR doThreeD2
   THENC
   inEmergeRp2 ← NEW!RECORD(emergeClass);
   IF (emergeClass:sonRp[inEmergeRp2] ← getIf(TRUE))
        = NULL!RECORD
   THEN BEGIN
        deleteRecord(inEmergeRp2);
        inEmergeRp2 ← NULL!RECORD;
        END;
   ENDC
ENDC
```

```
initPromptArea; !! defined in SAIL.AWN.;
ENDC
```

```
!! main loop;

IFC NOT BREADBOARD THENC  !! whole page;

WHILE TRUE DO BEGIN
   IF mailReceived THEN checkMailBox;
   !! use selection menu to find out which option to execute;
   actChar ← selMenu(mainMenu,lineNo,
      help,
      labelString,
      menutopLine,cmdCol,cmdCol - 2,
        IFC doEfl THENC 2 ELSEC -1 ENDC ,       !! menuMinLine;
        -1);                                    !! menuMaxLine;
   IF actChar = control("e") THEN DONE; !! control(x) defined in SAIL.AWN;
   restorePromptArea;
   !! counter, nextCounter, and the "do..." compiler switches assemble
      the following CASE loop according to whatever options are included;
   REDEFINE counter = 1;        !! defined in sail.awn;
   CASE lineNo OF BEGIN
IFC doEfl OR doDir OR doFDef    !! header line;
            THENC nextCounter;                                          ENDC
IFC doEfl   THENC [counter] BEGIN
                            menuInMrgFile(emergeInHead,inEmergeRp,defaultFileName,mrgExtensions);
                            !! input or choose a merge file, update main menu;
                            IF inEmergeRp = NULL!RECORD
                               THEN mainMenu[2] ← inputLine
                               ELSE mainMenu[2] ← inputLine & "("&
                                 (mrgMakeFileName(emergeClass:sonRp[inEmergeRp]))&")";
                             END;                           nextCounter; ENDC
IFC doDir    THENC [counter] makeDir(inEmergeRp);          nextCounter; ENDC
IFC doFDef   THENC [counter] makeFDefaults(inEmergeRp);    nextCounter; ENDC
IFC doEfl OR doDir OR doFDef
            THENC nextCounter; !! blank line;                           ENDC
IFC doShow OR doThreeD OR doThreeD2 OR doSpect OR doSlice !! header line;
            THENC nextCounter;                                          ENDC
IFC doShow   THENC [counter] makeshowOne(inEmergeRp);      nextCounter; ENDC
IFC doThreeD THENC [counter] makeThreeD(inEmergeRp);       nextCounter; ENDC
IFC doThreeD2 THENC [counter] mak2ThreeD(inEmergeRp, inEmergeRp2);     nextCounter; ENDC
IFC doSpect  THENC [counter] makeSpect(inEmergeRp);        nextCounter; ENDC
IFC doSlice  THENC [counter] makeSlice(inEmergeRp);        nextCounter; ENDC
IFC doShow OR doThreeD OR doSpect OR doSlice OR doThreeD2 !! blank line;
            THENC nextCounter;                                          ENDC
IFC doSplMer OR doEdSeg OR doedSeg2 OR doMfDif          !! header line;
            THENC nextCounter;                                          ENDC
IFC doSplMer THENC [counter] makSplMer(inEmergeRp);        nextCounter; ENDC
IFC doEdSeg  THENC [counter] makeEdSeg(InemergeRp);        nextCounter; ENDC
IFC doedSeg2 THENC [counter] make2EdSeg(InemergeRp,inEmergeRp2); nextCounter; ENDC
IFC doMfDif THENC [counter]   makMfDif(inEmergeRp); nextCounter; ENDC
IFC doSplMer OR doEdSeg OR doedSeg2 OR doMfDif
            THENC nextCounter; !! blank line;                           ENDC
IFC doInFun OR doOutFun         !! header line;
            THENC nextCounter;                                          ENDC
IFC doInFun  THENC [counter] makeinFun(inemergeRp);        nextCounter; ENDC
IFC doOutFun THENC [counter] makeseg(inemergeRp);          nextCounter; ENDC
IFC doInFun OR doOutFun         !! blank line;
            THENC nextCounter;                                          ENDC
IFC doNews OR doDebug           !! header line;
            THENC nextCounter;                                          ENDC
IFC doNews   THENC [counter] news; nextCounter; ENDC
IFC doDebug  THENC [counter] printFileList(emergeInHead); nextCounter; ENDC
      ELSE
            BEGIN
            restorePromptArea;
            OUTSTR(crlf&"YOU HAVE REACHED THE TWILIGHT ZONE ");
            END
      END; !! end of CASE lineNo;
   END;

ENDC
```

```
!! breadBoard version of EMERGE;

IFC breadBoard THENC !! whole page;
REQUIRE crlf&"not all features in breadBoard " MESSage;

STRING tmpStr;
RECORD!POINTER(emergeClass) inEmergeRp, emergeRp2;
dont_move ← TRUE;  !! see comment at beginning of file regarding this grnlib bug trap;

∂ initPromptArea; REQUIRE "
prompt area not initted " MESSAGE;

inEmergeRp ← NEW!RECORD(emergeClass);
REQUIRE crlf&"taking udp1:aa3l1a.mf[a,awn]" MESSAGE;
tmpStr ← "udp1:aa3l1a.mf[a,awn]";
OUTSTR("Trying to read "&tmpStr&" ");
emergeClass:sonRp[inEmergeRp] ← NEW!RECORD(mfClass);
REQUIRE crlf&"sonRp for default input file name NEW!RECORDed to mfClass " MESSAGE;
mrgClass:readAlter[emergeClass:sonRp[inEmergeRp]] ← FALSE;
getInMergeFile(tmpStr,mfType,emergeClass:sonRp[inEmergeRp],mrgExtensions,TRUE);

IFC doThreeD2 THENC

emergeRp2 ← NEW!RECORD(emergeClass);
tmpStr ← "udp2:ca3toc.my[gro,AWN]";
OUTSTR("Trying to read "&tmpStr&" ");
emergeClass:sonRp[emergeRp2] ← NEW!RECORD(mfClass);
REQUIRE crlf&"sonRp for default input file name NEW!RECORDed to mfClass " MESSAGE;
mrgClass:readAlter[emergeClass:sonRp[emergeRp2]] ← FALSE;
getInMergeFile(tmpStr,mfType,emergeClass:sonRp[emergeRp2],mrgExtensions,TRUE);
ENDC

IFC doSlice  THENC  makeSlice(inEmergeRp); ENDC
IFC doDir THENC makeDir(inEmergeRp); ENDC
IFC doSplMer THENC makSplMer(inEmergeRp); ENDC
IFC doInFun THENC makeInFun(inEmergeRp); ENDC
IFC doOutFun THENC makeSeg(inEmergeRp); ENDC
IFC doSpect THENC makeSpect(inEmergeRp); ENDC
IFC doShow THENC makeShowOnE(inEmergeRp); ENDC
IFC doThreed THENC makeThreeD(inEmergeRp); ENDC
IFC doThreed2 THENC mak2ThreeD(inEmergeRp, emergeRp2); ENDC
IFC doEdSeg THENC makeedSeg(inEmergeRp); ENDC
IFC doMfDif THENC makMfDif(inEmergeRp); ENDC

ENDC
```

```
END   "emerge";
```

File under: A Date: 22 Jan 1985 Name: John Strawn

Name: John Strawn

Project: J Programmer: AWN

File Name: UDP2:APPR.SAI[LIB,AWN]

File Last Written: 21:09 21 Jan 1985

Time: 13:00 Date: 22 Jan 1985

Center for Computer Research
in Music and Acoustics
Department of Music
Stanford, California

```
COMMENT ⊗   VALID 00013 PAGES
C REC  PAGE   DESCRIPTION
C00001 00001
C00002 00002     !! external declarations
C00004 00003     !! interface to approximation
C00005 00004     !! erToInt
C00006 00005     !! mic2Fun
C00012 00006     !! entry defs, PRELOAD for splMerMenu
C00017 00007     !! splMerMenuToRp, splMerRpToMenu
C00024 00008     !! initsplMerMenu
C00028 00009     !! xLabel, help
C00034 00010     !! controlGInit ampInit freqInit
C00039 00011     !! splMerControlG
C00051 00012     !! makSplMer
C00054 00013     END   "program"
C00055 ENDMK
C⊗;
```

```
COMMENT !! external declarations;

IFCR NOT DECLARATION(garply) THENC
ENTRY;
BEGIN "program";
REQUIRE "DSK:sail.awn[rom,awn]" SOURCE!FILE;
REQUIRE "DSK:grnlib.hdr[lib,bil]" SOURCE!FILE;
REQUIRE "DSK:awnlib.hdr[sub,sys]" SOURCE!FILE;
REQUIRE "DSK:menu.hdr[sub,sys]" SOURCE!FILE;
REQUIRE "DSK:merge.hdr[lib,awn]" SOURCE!FILE;
REQUIRE "DSK:emerge.hdr[lib,awn]" SOURCE!FILE;
ENDC

DEFINE debug = FALSE,
   includeEachStep = FALSE,  COMMENT currently broken --- pdl ov
                                from inside approx.  3/25/83;
   Trace = FALSE;

IFC NOT includeEachStep THENC
REQUIRE crlf&"turned off ""look at each step?"" " MESSAGE; ENDC

IFC debug THENC REQUIRE crlf&"debug turned on in splMer " MESSAGE; ENDC
IFC trace THENC REQUIRE crlf&"trace turned on in splMer " MESSAGE; ENDC
```

```
!! interface to approximation;

DEFINE Dont_Require_approx_Dammit = TRUE;
REQUIRE "dsk:PR.HDR[lib,AWN]" SOURCE!FILE;
REQUIRE "dsk:CPR.HDR[lib,AWN]" SOURCE!FILE;

INTERNAL RECORD!CLass micro(microFields);
```

```
!! erToInt;

SIMPLE INTEGER PROCEDURE erToInt(STRING errorString);

   CASE (upChar(stripSpaces(errorString)[1 FOR 1])) OF BEGIN
      ["I"] RETURN(integralErrorNorm);
      ["M"] RETURN(meanErrorNorm);
      ["X"] RETURN(maximumErrorNorm);
      ELSE BEGIN
         OUTSTR(crlf&"Can't decipher error norm;  defaulting to ""integral """);
         RETURN(integralErrorNorm);
         END
      END;
```

```
!! mic2Fun;

PROCEDURE mic2Fun(
   RECORD!POINTER(micro) microHead;
   INTEGER chanNo;
   BOOLEAN ampBool;
   RECORD!POINTER(emergeClass) emergeRp);

   BEGIN

   INTEGER nSegs, segCnt;
   RECORD!POINTER(ANY!CLASS) segFunRp, microTail, microRp;
   IFC trace THENC OUTSTR("entering mic2Fun"&crlf); ENDC
   IF microHead = NULL!RECORD THEN RETURN;
   OUTSTR(crlf&"Storing approximation inside A slot ");
   segFunRp ← emergeClass:splMerFuncs[emergeRp][chanNo,(IF ampBool THEN dirAmpType ELSE dirFreqType)];


   IF segFunRp NEQ NULL!RECORD
      THEN BEGIN
      deleteRecord(segFunRp);
      END;
   nSegs ← countList(microHead)+1;
   IF micro:xBeg[microHead] NEQ 0 THEN nSegs ← nSegs+1;
   getTail(microHead,microTail);
   IF mrgHiPnt(mrgRp) > micro:xEnd[microTail] THEN nSegs←nSegs+1;
   segFunRp ← makenewseg (nSegs);
   seg:name[segFunRp] ← (IF ampBool THEN "A" ELSE "F")&CVS(chanNo);

   segCnt ← 1;
   seg:minTime[segFunRp] ← +100000000;
   seg:maxTime[segFunRp] ← -100000000;
   seg:maxVal[segFunRp] ← seg:minVal[segFunRp] ← micro:yBeg[microHead];
   IF micro:xBeg[microHead] NEQ 0
      THEN BEGIN
      IFC debug THENC OUTSTR(crlf&"forcing segCnt = "&CVs(segCnt)&" "); ENDC
      seg:times[segFunRp][segCnt] ← 0;
      seg:values[segFunRp][segCnt] ← (IF ampBool THEN 0 ELSE
         mfGetWd1(mrgRp,chanNo,dirFreqType));
      segCnt ← segCnt + 1;
      END;

   microRp ← microHead;
   IFC debug THENC OUTSTR(crlf&"creating segCnt = "); ENDC
   WHILE microRp NEQ NULL!RECORD DO
      BEGIN
      IFC debug THENC OUTSTR(CVS(segCnt)&" "); ENDC
IFC debug THENC OUTSTR(CVOS(rpToInt(microRp))&" "); ENDC
      seg:times[segFunRp][segCnt] ← pointToSec(micro:xBeg[microRp],mrgRp);
      seg:values[segFunRp][segCnt] ← micro:yBeg[microRp];
IFC debug THENC OUTSTR(CVf(micro:yBeg[microRp])&" "&
   CVF(seg:values[segFunRp][segCnt])&crlf); ENDC
        !! this works for both freq and amplitude! --- update was called from splmer;
      seg:maxVal[segFunRp] ← seg:maxval[segFunRp] MAX seg:values[segFunRp][segCnt];
      seg:minVal[segFunRp] ← seg:minval[segFunRp] MIN seg:values[segFunRp][segCnt];
      segCnt ← segCnt + 1;
      microRp ← micro:next[microRp];
      END;

   IFC debug THENC OUTSTR(crlf&"final segCnt = "&CVS(segCnt)&" "); ENDC
   seg:times[segFunRp][segCnt] ← pointToSec(micro:xEnd[microTail],mrgRp);
   seg:values[segFunRp][segCnt] ←
      (IF ampbool
       THEN micro:yEnd[microTail]
       ELSE seg:values[segFunRp][segCnt] ← mfGetWd1(mrgRp,chanNo,dirFreqType));

   seg:maxVal[segFunRp] ← seg:maxval[segFunRp] MAX seg:values[segFunRp][segCnt];
   seg:minVal[segFunRp] ← seg:minval[segFunRp] MIN seg:values[segFunRp][segCnt];
```

```
   IF mrgHiPnt(mrgRp) > micro:xEnd[microTail]
      THEN BEGIN
      segCnt ← segCnt + 1;
      IFC debug THENC OUTSTR(crlf&"forcing segCnt = "&CVs(segCnt)&" "); ENDC
      seg:times[segFunRp][segCnt] ← pointToSec(mrgHiPnt(mrgRp),mrgRp);
      seg:values[segFunRp][segCnt] ←
         (IF ampbool
          THEN micro:yEnd[microTail]
          ELSE seg:values[segFunRp][segCnt] ← mfGetWd1(mrgRp,chanNo,dirFreqType));
      seg:maxVal[segFunRp] ← seg:maxval[segFunRp] MAX seg:values[segFunRp][segCnt];
      seg:minVal[segFunRp] ← seg:minval[segFunRp] MIN seg:values[segFunRp][segCnt];
      END;

   seg:minTime[segFunRp] ← seg:times[segFunRp][1];
   seg:maxTime[segFunRp] ← seg:times[segFunRp][segCnt];

   emergeClass:splMerFuncs[emergeRp]
        [chanNo,(IF ampBool THEN dirAmpType ELSE dirFreqType)] ← segFunRp;
∂  linkFuns(emergeClass:splMerFuncs[emergeRp]);
   IFC trace THENC OUTSTR("leaving  mic2Fun"&crlf); ENDC
   END;
```

```
!! entry defs, PRELOAD for splMerMenu;

!! The following is inspired by TEX.  For each counter/nextCounter
pair, we simply advance counter by 1.  These defines thus set up
constants which index into the first dimension of array splMerMenu. Obviously,
if you change the order of entries in the PRELOAD statement below, you
must change the order of these DEFINEs accordingly;

REDEFINE counter = 1;
DEFINE firstsplMerEntry = counter;

DEFINE chanNoEntry = counter;  nextCounter;
DEFINE ampBoolEntry = counter; nextCounter;
   nextCounter; !! explanatory line;
DEFINE errorNormEntry = counter; nextCounter;
   nextCounter; !! explanatory line;
DEFINE thresholdEntry = counter; nextcounter;
IFC includeEachStep THENC DEFINE displayEntry = counter; nextCounter; ENDC
DEFINE tBegEntry = counter; nextCounter;
DEFINE tEndEntry = counter; nextcounter;
DEFINE outNameEntry = counter; nextCounter;
DEFINE debugEntry = counter; nextCounter;
DEFINE xLabelEntry  = counter; nextCounter;
DEFINE includeLabelEntry = counter; nextCounter;
DEFINE pltNameEntry = counter;  nextCounter;
DEFINE linesEntry = counter; nextCounter;
nextCounter;
nextCounter;
DEFINE dBEntry = counter; nextCounter;
DEFINE dbRangeEntry = counter; nextCounter;
nextCounter;
nextCounter;
DEFINE channelOverlapEntry = counter; nextCounter;
DEFINE squelchEntry = counter; nextCounter;
DEFINE squelchFreqEntry = counter; nextCounter;
DEFINE squelchDbEntry = counter; nextCounter;

DEFINE lastsplMerEntry = counter-1,
   indentLength = 3,
   longestsplMerEntry = squelchEntry,
   chanNoLine =    <("Channel No.")>,
   endTimeLine =   <("End time")>;     COMMENT modified at run time;

PRELOAD!WITH
chanNoLine,"",intTyped,
"Approximate amplitude function?","",boolTyped,
"    (FALSE = use frequency function)","",blankLineTyped,
"Error Norm","",ucTyped,
"    (I = integral, X = maximum, M = mean)","",blankLineTyped,
"Threshold","",realTyped,
IFC includeEachStep THENC "Display each step? (T or F)","",boolTyped, ENDC
"Beg time",""    ,numTyped,
endTimeLine,""   ,numTyped,
"Output File Name","",ucTyped, !! so "(" kludge will work;
"Debug  File Name","",ucTyped, !! so "(" kludge will work;
"Label","",untyped,
"Include label (T or F)","",boolTyped,
"Plot File Name","",ucTyped,
"Show merge function in Lines? (F = dots)","",boolTyped,
"","",blankLineTyped,
"For amplitude plots:","",blankLineTyped,
"  Use dB for amp plot? (F = linear, T = dB)","",boolTyped,
"  dB Range","",realTyped,
"","",blankLineTyped,
"For frequency functions:","",blankLineTyped,
"  Freq. Channel Overlap (0.0 to 1.0)","",realTyped,
"  Squelch frequencies before approximating? (T or F)","",boolTyped,
"     Squelch frequencies to this frequency:","",realTyped,
"     When the amplitude is this many dB down:","",realTyped;
```

```
mySAFE STRING ARRAY splMerMenu[firstSplMerEntry:lastSplMerEntry,menuCmd:menuType];
```

```
!! splMerMenuToRp, splMerRpToMenu;

PROCEDURE splMerMenuToRp(RECORD!POINTER(emergeClass) emergerp);
   BEGIN
   INTEGER brk;
   IFC trace THENC OUTSTR("entering splMerMenuToRp"&crlf); ENDC
   splMerclass:lines[splMerRp]  ← boolTruefalse(splMerMenu[linesEntry, menuVal]);
   splMerClass:channelOverlap[splMerRp] ← REALSCAN(splMerMenu[ChannelOverlapEntry,menuVal],brk);
   splMerClass:xLabel [splMerRp] ← splMerMenu[xLabelEntry, menuVal];
   splMerClass:squelch[splMerRp] ← boolTrueFalse(splMerMenu[xLabelEntry, menuVal]);
   splMerClass:squelchFreq[splMerRp]  ← REALSCAN(splMerMenu[squelchFreqEntry, menuVal],brk);
   splMerClass:squelchdbRange[splMerRp]  ← REALSCAN(splMerMenu[squelchdBEntry, menuVal],brk);
   splMerClass:lines  [splMerRp] ← boolTrueFalse(splMerMenu[linesEntry, menuVal]);
   splMerClass:includeLabel[splMerRp] ← boolTrueFalse(splMerMenu[includeLabelEntry,menuVal]);
   splMerClass:pltName[splMerRp] ← splMerMenu[pltNameEntry,menuVal];
   splMerClass:dbRange[splMerRp] ← REALSCAN(splMerMenu[dbRangeEntry,  menuVal],brk);
   splMerClass:db[splMerRp] ← boolTrueFalse(splMerMenu[dbEntry, menuVal]);
   splMerClass:outName[splMerRp] ← splMerMenu[outNameEntry,menuVal];
   splMerClass:threshold[splMerRp] ← REALSCAN(splMerMenu[thresholdEntry,menuVal],brk);
   splMerClass:errorNorm[splMerRp] ← erToInt(splMerMenu[errorNormEntry,menuVal]);
   splMerClass:chanNo[splMerRp] ← INTSCAN(splMerMenu[chanNoEntry,menuVal],brk);
   splMerClass:tBeg   [splMerRp] ← splMerMenu[tbegEntry,   menuVal];
   splMerClass:tEnd   [splMerRp] ← splMerMenu[tEndEntry,   menuVal];
   splMerClass:ampBool[splMerRp] ← boolTrueFalse(splMerMenu[ampBoolEntry,menuVal]);
IFC includeEachStep THENC
   splMerClass:display[splMerRp] ← boolTrueFalse(splMerMenu[displayEntry,menuVal]);
ENDC
   IFC trace THENC OUTSTR("leaving  splMerMenuToRp"&crlf); ENDC
   END;

PROCEDURE splMerRpToMenu(RECORD!POINTER(emergeClass) emergeRp);
   BEGIN
   STRING tmpStr;
   IFC trace THENC OUTSTR("entering splMerRpToMenu"&crlf); ENDC
   splMerMenu[channelOverlapEntry  ,menuVal] ← stripSpaces(CVf(splMerClass:channelOverlap[splMerRp]))
;
   splMerMenu[dbEntry,  menuVal] ← trueFalseString(splMerClass:db     [splMerRp]);
   splMerMenu[linesEntry,menuVal] ← trueFalseString(splMerclass:lines[splMerRp]);
   splMerMenu[squelchEntry,  menuVal] ← trueFalseString(splMerClass:squelch[splMerRp]);
   splMerMenu[squelchFreqEntry,menuVal] ← CVF(splMerClass:squelchFreq[splMerRp]);
   splMerMenu[squelchdBEntry,menuVal] ← CVF(splMerClass:squelchdbRange[splMerRp]);
   splMerMenu[linesEntry,  menuVal] ← trueFalseString(splMerClass:lines  [splMerRp]);
   splMerMenu[dbRangeEntry,  menuVal] ← stripSpaces(CVF(splMerClass:dbRange[splMerRp]));
   splMerMenu[outNameEntry,menuVal] ←     splMerClass:outName[splMerRp] ;
IFC includeEachStep THENC
   splMerMenu[displayEntry,menuVal] ← trueFalseString(splMerClass:display[splMerRp]);
ENDC
   splMerMenu[ampBoolEntry,menuVal] ← trueFalseString(splMerClass:ampBool[splMerRp]);
   splMerMenu[thresholdEntry,menuVal] ← CVF(splMerClass:threshold[splMerRp]);
   CASE splMerClass:errorNorm[splMerRp] OF BEGIN
      [integralErrorNorm] splMerMenu[errorNormEntry,menuVal] ← "I";
      [maximumErrorNorm] splMerMenu[errorNormEntry,menuVal] ← "X";
      [meanErrorNorm] splMerMenu[errorNormEntry,menuVal] ← "M";
      ELSE splMerMenu[errorNormEntry,menuVal] ← "I"
      END;
   splMerMenu[chanNoEntry,menuVal] ← CVS(splMerClass:chanNo[splMerRp]);
   splMerMenu[tBegEntry   ,menuVal] ← splMerClass:tBeg   [splMerRp];
   splMerMenu[tEndEntry   ,menuVal] ← splMerClass:tEnd   [splMerRp];
   splMerMenu[includeLabelEntry,menuVal] ← trueFalseString(splMerClass:includeLabel[splMerRp]);
   splMerMenu[pltNameEntry,menuVal] ←splMerClass:pltName[splMerRp] ;
   splMerMenu[xLabelEntry ,menuVal] ←     splMerClass:xLabel [splMerRp] ;
   IFC trace THENC OUTSTR("leaving  splMerRpToMenu"&crlf); ENDC
   END;
```

```
!! initsplMerMenu;

PROCEDURE initsplMerMenu (RECORD!POINTER(emergeClass) emergeRp; STRING nameWithoutExtension);
   BEGIN
   INTEGER brk;
   IFC trace THENC OUTSTR("entering initsplMerMenu"&crlf); ENDC
   IF splMerRp = NULL!RECORD
      THEN BEGIN
         splMerRp ← NEW!RECORD(splMerClass);
         pmClearVals(splMerMenu);
         splMerMenu[channelOverlapEntry,menuVal] ← "1.0";
         splMerMenu[linesEntry,menuVal] ← "TRUE";
         splMerMenu[squelchFreqEntry,menuVal] ← CVF(mfGetWd1(mrgRp,
            1,dirFreqType));
         splMerMenu[pltNameEntry, menuVal] ←
            "("&mrgClass:dev[mrgRp]&":"&nameWithoutExtension&".C/Q"&
                CVS(mfClass:begChan[mrgRp])&")          ";
         splMerMenu[includeLabelEntry,menuVal] ← "TRUE";
         splMerMenu[linesEntry,menuVal] ← "TRUE";
         splMerMenu[squelchEntry,menuVal] ← "TRUE";
         splMerMenu[xLabelEntry, menuVal] ← mrgClass:fileName[mrgrp]&"";
         splMerMenu[squelchDbEntry,menuVal] ←
         splMerMenu[dbRangeEntry,menuVal] ← "50";
         splMerMenu[dBEntry,menuVal] ← "FALSE";
         splMerMenu[chanNoEntry,menuVal] ← CVS(mfClass:begChan[mrgRp] MAX 1);
         splMerMenu[tBegEntry,    menuVal] ← "0.0";
         splMerMenu[tEndEntry,    menuVal] ← stripSpaces(CVF(
            pointToSec(mrgHiPnt(mrgrp),mrgRp)));
IFC includeEachStep THENC
         splMerMenu[displayEntry,menuVal] ← "FALSE";
ENDC
         splMerMenu[ampBoolEntry,menuVal] ← "TRUE";
         splMerMenu[errorNormEntry,menuVal] ← "I";
         splMerMenu[thresholdEntry,menuVal] ← "10000";
         splMerMenu[outNameEntry, menuVal] ←
            "("&mrgClass:dev[mrgRp]&":"&nameWithoutExtension&".SMF/Q)          ";
         END
      ELSE BEGIN
         splMerRpToMenu(emergeRp);
         END;
   !! debugEntry isn't in the record pointer;
   splMerMenu[debugEntry, menuVal] ← "("&"DSK:"&nameWithoutExtension&".TMP/Q)       ";
   splMerMenu[chanNoEntry,menuCmd] ← chanNoLine&" ("&
      CVS(mfClass:begchan[mrgRp])&":"&
      CVS(mfClass:endchan[mrgRp])&")";
   splMerMenu[tEndEntry,    menuCmd] ← endTimeLine&" ("&
      stripSpaces(CVF(pointToSec(mrgHiPnt(mrgrp),mrgRp)))[1 TO 4]&")";
   IFC trace THENC OUTSTR("leaving  initsplMerMenu"&crlf); ENDC
   END;
```

```
!! xLabel, help;

REQUIRE "{}{}" DELIMITERS;
      DEFINE xLabel = {
      (IF splMerMenu[xLabelEntry,menuVal] = ""
       THEN ""
       ELSE splMerMenu[xLabelEntry,menuVal]&" ")&
       (getDate)&" "&(getTim)&
      " Channel "&
      CVS(chanNo)&crlf&
      (IF (boolTrueFalse(splMerMenu[ampBoolEntry,menuVal]))
          THEN (IF (boolTrueFalse(splMerMenu[dbEntry,menuVal]))
                THEN "(dB) "
                ELSE "(linear) ")
          ELSE "")&"Threshold = "&CVF(threshold)&" "&
       (CASE errInt OF ("Maximum","Integral","Mean"))
       &" Error Norm; Approximated by "&CVS(microLen)&" segment"&
         (IF microLen > 1 THEN "s " ELSE " ")};
REQUIRE UNSTACK!DELIMITERS;

   !! XJFILLED on an entry-by-entry basis to 0,3,82;
   DEFINE help = <(
"You can invoke Pavlidis' split/merge algorithm to approximate the functions in
the file.  See J. Strawn, ""Approximation and Syntactic Analysis..."", Computer
Music Journal 4(3), 1980, Section 4.2.2, p. 9.

"&chanNoLine&
          " --- The channel of the phase vocoder output to be approximated.
   The numbers in parentheses give the channels available in this
   file (="&CVS(mfClass:begChan[mrgRp])&":"&
   CVS(mfClass:endChan[mrgRp])&").

"&splMerMenu[tBegEntry,menuCmd]&
    " --- Begin time, in seconds.  We assume the file starts
    at 0.0.  You may also type #nnn to get sample number
    nnn.  Note that this is the sample in the merge file,
    not the sample number in the original sound file.

"&endTimeLine&" --- End time to be displayed.  The number in parentheses
    gives the duration of the file (="&CVF(pointToSec(mrgHiPnt(mrgRp),mrgRp))
    &").
    You may also type #nnn for a sample number, or ""∞"" to
    get the end value.

"&
IFC includeEachStep THENC
splMerMenu[displayEntry,menuCmd]&
                        " --- TRUE means that you will be shown a plot of
   the approximation at every step (split, merge, adjust) along the way.
   A nice show, useful if you haven't seen the algorithm work before.
"& ENDC

splMerMenu[errorNormEntry,menuCmd]&
          " --- Pick one. See the CMJ article, Section 3, p. 6.

"&splMerMenu[thresholdEntry,menuCmd]&
          " --- The larger this number, the looser the approximation.  This is the
   ""threshold"" in Fig. 5 of the Computer Music Journal article, p. 10.

"&splMerMenu[debugEntry,menuCmd]&
                " --- Write out a text file containing a blow-by-blow descrip-
   tion of each step of the approximation.  Probably useful only to JAWN.

"&splMerMenu[outNameEntry,menuCmd]&
                " --- Write all of the approximated functions available for
   this input file to a .FUN file.  Typically you will write out this file once,
   after you've created approximations for all of the functions.

"&splMerMenu[xLabelEntry,menuCmd]&
```

```
   " --- You can supply a string here to appear as a label on the plot.

"&splMerMenu[squelchEntry,menuCmd][3 TO INF]&
                        " --- TRUE means to slam frequencies to the center
   frequency specified when the amplitude falls below the range specified. The
   dB are measured relative to the maximum of the amplitude in this channel.

Channel overlap here affects only the display once the approximation is
   done;  it does not affect the values passed to the approximation routine.

File Names:  As long as the file name in the menu begins with ""("", the file will
   not be written.  We supply a default file name in parentheses for you.  Just
   remove the first parenthesis when you want to write out the functions, and
   edit the file name as you wish.

After the approximation is finished, you will be returned to this menu.
To abort the approximation process, type Esc-I."

)>;
```

```
!! controlGInit ampInit freqInit;

PROCEDURE controlGInit (
        RECORD!POINTER (mfClass) rp;
        REFERENCE RECORD!POINTER (winClass) winRp;
        REFERENCE REAL threshold;
        REFERENCE BOOLEAN  showMerge, ampBool, mrgInDots;
        REFERENCE INTEGER chanNo, errInt, displaySkipPts,
            mfBegPt, mfEndPt);

    BEGIN
    INTEGER brk;
    !! general initialization. Just do win1 here;
    winRp← NEW!RECORD(winClass);
    threshold ← REALSCAN(splMerMenu[thresholdEntry,menuVal],brk);
    MrgInDots ← NOT boolTrueFalse(splMerMenu[linesEntry,menuVal]);
    errInt ← (erToInt(splMerMenu[errorNormEntry,menuVal]));
    showMerge ← TRUE;
    ampBool ← boolTrueFalse(splmerMenu[ampBoolEntry,menuVal]);

    winClass:xVal0[winRp]  ← numbTime(splmerMenu[tBegEntry,menuVal],rp);
    winClass:xVal1[winRp] ← infNumbTime(splmerMenu[tEndEntry,menuVal],rp);
    IFC debug THENC
    OUTSTR("winClass:xVal0[winRp] "&CVF(winClass:xVal0[winRp])&"xVal1 "&CVF(winClass:xVal1[winRp])&crl
f); ENDC
    chanNo ←
    ((INTSCAN(splmerMenu[chanNoEntry,menuVal],brk) MAX
        mfClass:begChan[rp]) MIN
        mfClass:endChan[rp]);
    IFC debug THENC
        OUTSTR("xVal0 "&CVF(winClass:xVal0[winRp])&
        "xVal1 "&CVF(winClass:xVal1[winRp])&crlf); ENDC
    mfEndPt ← mfSecToPoint(winClass:xVal1[winRp],rp) MIN (mfHiPnt(rp));
    mfBegPt ← (mfSecToPoint(winClass:xVal0[winRp],rp) MAX mrgLoPnt) MIN mfEndPt;
    displaySkipPts ← ((mfEndPt-mfBegPt+1) DIV 512) MAX 1;
    END;

PROCEDURE ampInit(
        RECORD!pOINTER (emergeClass) emergeRp;
        INTEGER chanNo, mfBegPt, mfEndPt;
        REFERENCE BOOLEAN dbBool, useEachMax;
        REFERENCE REAL maxAmp, dbRange);

    BEGIN
    INTEGER brk;
    IFC trace THENC OUTSTR("entering ampInit "&crlf); ENDC
    dbBool ← boolTrueFalse(splmerMenu[dbEntry,menuVal]);
    IF dbBool
        THEN BEGIN
        dbRange ← REALSCAN(splmerMenu[dbRangeEntry,menuVal],brk);
        END;
    maxAmp ← getMyMax(mfBegPt, mfEndPt, chanNo, mrgrp);
    IFC trace THENC OUTSTR("leaving ampInit "&crlf); ENDC
    END;

PROCEDURE freqInit(
        RECORD!POINTER (emergeClass) emergeRp;
        INTEGER chanNo, mfBegPt, mfEndPt;
        REFERENCE BOOLEAN squelchBool;
        REFERENCE REAL hiFreq, loFreq, squelchDbRange, squelchAmp, squelchFreq,
            channelOverlap);

    BEGIN
    INTEGER brk;
    REAL  cFreq;
    IFC trace THENC OUTSTR("entering freqInit "&crlf); ENDC
    squelchBool ← boolTrueFalse(splmerMenu[squelchEntry,menuVal]);
    channelOverlap ← REALSCAN(splmerMenu[channelOverlapEntry,menuVal],brk);
    cFreq ← mfClass:clock[mrgrp]*ChanNo/mfClass:N[mrgRp];
```

```
    loFreq ← cFreq - cFreq*channelOverlap;
    hiFreq ← cFreq + cFreq*channelOverlap;

    IF squelchBool THEN
        BEGIN
        squelchDBRange ← REALSCAN(splmerMenu[squelchdBEntry, menuVal],brk);
        squelchAmp ←  getMyMax(mfBegPt,mfEndPt,chanNo,mrgRp);
        squelchFreq ← REALSCAN(splmerMenu[squelchFreqEntry,menuVal],brk);
        END;

    IFC trace THENC OUTSTR("leaving freqInit "&crlf); ENDC
    END;
```

```
!! splMerControlG;

PROCEDURE splMerControlG(RECORD!POINTER(emergeClass) emergeRp);

    BEGIN
    INTEGER errInt,brk, segBegPt, segEndPt, outChan,
        debugChan, microLen, chanNo, displaySkipPts,
        mfBegPt, mfEndPt;
    REAL threshold,  maxAmp, dbRange,
        hiFreq, loFreq, squelchDbRange, squelchAmp,
        squelchFreq, channelOverlap;
    BOOLEAN MrgInDots,   showMerge,  ampBool,  dbBool, useEachMax, squelchBool;
    RECORD!POINTER(micro) microHead, tmpRp;
    RECORD!POINTER (seg) segFunRp;
    RECORD!POINTER(winClass) winRp;
    INTEGER ARRAY labelBuf[0:LENGTH(splMerMenu[xLabelEntry,menuVal])+500];
    mySAFE STRING ARRAY savesplMerMenu[
        ARRINFO(splMerMenu,1):ARRINFO(splMerMenu,2),
        ARRINFO(splMerMenu,3):ARRINFO(splMerMenu,4)];

    iniEscI;
    escISeen ← FALSE;
    clearScreen;
    initPromptArea;
    restorePromptArea;
    IFC trace THENC OUTSTR("entering splMerControlG"&crlf); ENDC
    ARRTRAN(savesplMerMenu,splMerMenu);  !! to overcome the effects of INTSCAN, below);
    OUTSTR(crlf&"Initialization ... ");

    !! do we believe the file names?;
    IF NOT cTryOutFileName(splMerMenu,saveSplMerMenu,outNameEntry,outchan)
        THEN RETURN;
    IF NOT cTryOutFileName(splMerMenu,saveSplMerMenu,debugEntry,debugChan)
        THEN RETURN;
    IF debugChan NEQ unopenedChannel
        THEN BEGIN
        OUTSTR(crlf&"debug currently disabled, sorry ");
        CLOSE(debugChan);
        RELEASE(debugChan);
        debugChan ← unopenedChannel;
        END;

    !! general initialization;
    controlGInit (mrgrp, winRp, threshold, showMerge,
        ampBool, mrgInDots, chanNo, errInt, displaySkipPts,
        mfBegPt, mfEndPt);
    IF ampBool
    THEN ampInit( emergeRp, chanNo, mfBegPt, mfEndPt, dbBool,
        useEachMax, maxAmp, dbRange)
    ELSE freqInit( emergeRp, chanNo, mfBegPt, mfEndPt,
        squelchBool, hiFreq, loFreq, squelchDbRange, squelchAmp,
        squelchFreq, channelOverlap);

    !! munch the function;
    BEGIN "dummy block"
    mySAFE REAL ARRAY inFunc[mfBegPt:mfEndPt];
    IF (mfRdFun(mrgRp,inFunc,
            chanNo,
            (IF ampBool THEN dirAmpType ELSE dirFreqType)))
    THEN OUTSTR(crlf&"Can't find "&(IF ampBool THEN "amp" ELSE "frequency")&
        " function for Channel "&CVS(chanNo)&
        " at time "&CVF(pointToSec(mfBegPt,mrgRp))&" ")
        !! and basically go back where you came from;
    ELSE BEGIN !! the real work's here;
        IF (NOT ampBool) AND boolTrueFalse(splMerMenu[squelchEntry,menuVal])
        THEN squelch(mrgRp, chanNo, squelchDbRange, squelchFreq, squelchAmp,
            inFunc, 1);
        !! set up list of approximated points;
        microHead ← NEW!RECORD(micro);
```

```
        micro:xBeg[microHead] ← mfBegPt;
        micro:xEnd[microHead] ← mfEndPt;
        !! do approximation;
        IFC includeEachStep THENC
            IF NOT boolTrueFalse(splMerMenu[displayEntry,menuVal]) THEN  ENDC
        OUTSTR(" WORKING ... (type Esc-I to abort) ");
        eachSplitMerge(
            microHead,
            inFunc,
            mrgClass:clock[mrgRp],
            mrgClass:compr[mrgRp],
            errInt,
            threshold,
            debugChan,
            IFC includeEachStep
            THENC
            boolTrueFalse(splMerMenu[displayEntry,menuVal])
            ELSEC    FALSE
            ENDC
            );
        IF debugChan NEQ unopenedChannel THEN BEGIN CLOSE(debugChan); RELEASE(debugChan); END;
        IFC includeEachStep THENC
            IF NOT boolTrueFalse(splMerMenu[displayEntry,menuVal]) THEN ENDC
        OUTSTR("Done ");
        IF NOT escISeen
        THEN BEGIN
            !! convert approximated function to SEG function;
            microLen ← countList(microHead);
            !! ... and store it in the splMerFuncs array;
            mic2Fun(microHead,chanNo,ampBool,emergeRp);
            segBegPt ← 1;
            segEndPt ← microLen;
            END;
            !! get rid of original form of approximation;
        tmpRp ← microHead;
        WHILE microHead NEQ NULL!RECORD DO
            BEGIN
            microHead ← micro:next[microHead];
            delRec(tmpRp,FALSE);
            tmpRp ← microHead;
            END;

        !! output updated approximated functions?;
        IF outChan NEQ unOpenedChannel
            THEN BEGIN
            IF NOT escISeen
            THEN wrtFuncFile(emergeClass:splMerFuncs[emergeRp],
                    outChan,
                    -1,-1, !! begChan, endChan;
                    TRUE, TRUE,         !! ampSide, freqSide;
                    -1,-1,      !! begTime, endTime;
                    0.0,        !! shift by begin time;
                    emergeClass:ignoredAFuncs[emergeRp],
                    "Functions APPROXIMATED"&crlf&"with Pavlidis algorithm from "&crlf&
                    mrgMakeFileName(mrgRp)
                        &(IF mrgClass:commentStr[mrgRp] = ""
                          THEN ""
                          ELSE crlf&mrgClass:commentStr[mrgRp]),
                    (IF fileDefaultRp = NULL!RECORD
                     THEN ""
                     ELSE fileDefaultClass:funcName[fileDefaultRp]),
                    saveSplMerMenu[outNameEntry,menuVal]);
            CLOSE(outChan); RELEASE(outChan);
            END;

        IF NOT escISeen
            THEN BEGIN
            !! show original with approximation;
            OUTSTR(" setting up display ... ");
```

```
            segFunRp ← emergeClass:splMerFuncs[emergeRp][chanNo,
                (IF ampBool THEN dirAmpType ELSE dirFreqType)];
            newArray(<INTEGER>,<winClass:buf[winRp]>,
                <[0:(((mfEndPt-mfBegPt+1)*2)
                 /displaySkipPts)+750+seg:nsegs[segFunRp]*2]>);
            initWindow(winClass:buf[winRp],screenXMin, screenYMin,
                screenXMax, screenYMax,
                FALSE,FALSE,FALSE); !! labelled, framed, opaque;

            IF ampBool
            THEN mfAShow(showMerge, mrgInDots, dbBool, TRUE,     !! useEachMax;
                chanNo, mfBegPt, mfEndPt, displaySkipPts,
                segBegPt, segEndPt,
                maxAmp, dbRange, winRp, mrgrp,
                segFunRp)
            ELSE mfFrShow(showMerge, mrgInDots, squelchBool, chanNo,
                mfBegPt, mfEndPt, displaySkipPts,
                segBegPt, segEndPt,
                channelOverlap,
                squelchDbRange,
                squelchFreq, squelchAmp, winRp, mrgrp, segFunRp);

            !! draw label too?;
            IF (boolTrueFalse(splMerMenu[includeLabelEntry,menuVal]))
            THEN heyMabelThatLabel(labelBuf,xLabel);
            refreshScreen;
            pltPrompt(parStrip(splMerMenu[PltNameEntry,menuVal]));
            !! output plot file;
            plotAwn(splMerMenu,savesplMerMenu,PltNameEntry);
            !! done with show and tell;
            relWindow(winClass:buf[winRp]);
            relWindow(labelBuf);
            END;
        END;
    END   "dummy block";

    !! clean house;
    ARRTRAN(splMerMenu,savesplMerMenu);
    IF winRp NEQ NULL!RECORD THEN deleteRecord(winRp);
    refreshScreen;
    IFC trace THENC OUTSTR("leaving  showOneControlG"&crlf); ENDC
    END;
```

```
!! makSplMer;

INTERNAL PROCEDURE makSplMer(RECORD!POINTER(emergeClass) emergeRp);
   BEGIN
   INTEGER actChar, lineNo;
   STRING commandFileName, nameWithoutExtension;

   IFC trace THENC OUTSTR("entering makSplMer"&crlf); ENDC
   IF NOT emrgConfirm(emergeRp,MFType) THEN RETURN;

   nameWithoutExtension ← justName(mrgmakeFileName(mrgRp));
   initsplMerMenu(emergeRp,nameWithoutExtension);
   lineNo ← 1;
   commandFileName ← nameWithoutExtension&".SMC";
        !! Steve McAdams Memorial Extension;

   !! make sure there's an array there for stuffing approximations into;
   ensSplMerArray(emergeRp);

   WHILE TRUE DO
      BEGIN
      actChar ← parMenu(splMerMenu,lineNo,commandFileName,
         help,
         "Parameters for appying split/merge to one channel from "&crlf&"  "&
            (whichMrgType(mrgClass:type[mrgrp]))&" file "&
            (mrgMakeFileName(mrgrp))&crlf&crlf,TRUE);
      IF actChar = control("e")
         THEN BEGIN
         splMerMenuToRp(emergeRp);
         pmClearVals(splMerMenu);
         IFC trace THENC OUTSTR("leaving  makSplMer"&crlf); ENDC
         RETURN ;
         END
      ELSE IF actChar = control("g") THEN
         BEGIN
         splMerControlG(emergeRp);
         END;
      END;
   END;
```

```
END   "program";
```