Name: Gareth Loy

Project: 1 Programmer: DGL

File Name: MBOX.SAI[SAM,DGL]

File Last Written: 14:33 15 Nov 1977

Time: 22:00 Date: 16 Nov 1977

Stanford University
Artificial Intelligence Laboratory
Computer Science Department
Stanford, California

```
COMMENT ⊗   VALID 00016 PAGES
C REC  PAGE   DESCRIPTION
C00001 00001
C00003 00002    begin "MBOX"
C00007 00003    ∂ Init. break tables
C00012 00004    ∂ Endgame
C00013 00005    ∂ record handling routines
C00021 00006    ∂ Reserved word and Variable lookup and allocation
C00033 00007    ∂ INSCAL, figures out whom to call
C00040 00008    ∂ DSK and TTY input routines
C00047 00009    ∂ SCANCALL, the main scan-parse loop
C00055 00010    ∂ SCANCALL parser
C00060 00011    ∂ SCANCALL stack manipulation
C00064 00012    ∂ SCANCALL scan routines
C00067 00013    ∂ SCANCALL reserved word scan procedures
C00071 00014    ∂ SCANCALL reserved word scan procedures continued
C00076 00015    ∂ SCANCALL main scan loop
C00084 00016    ∂ main prog
C00089 ENDMK
C⊗;
```

```
begin "MBOX"
require "MBOX.HDR[SAM,DGL]" source_file;

require "SAMTBL.SAI[SAM,DGL]" source_file;

require "SAMINS.REL[SAM,DGL]" load_module;
external recursive procedure INS0(Record_Pointer(callList) Fn;
        Record_Pointer(arrRec) Pns);
external recursive procedure INS1(Record_Pointer(callList) Fn;
        Record_Pointer(arrRec) Pns);
external recursive procedure INS2(Record_Pointer(callList) Fn;
        Record_Pointer(arrRec) Pns);
external recursive procedure INS3(Record_Pointer(callList) Fn;
        Record_Pointer(arrRec) Pns);
external recursive procedure INS4(
        ∂ Record_Pointer(callList) Fn;
        Record_Pointer(arrRec) Pns);
external recursive procedure INS5(Record_Pointer(callList) Fn;
        Record_Pointer(arrRec) Pns);
external recursive procedure INS6(Record_Pointer(callList) Fn;
        Record_Pointer(arrRec) Pns);
external recursive procedure INS7(Record_Pointer(callList) Fn;
        Record_Pointer(arrRec) Pns);
external recursive procedure INS8(Record_Pointer(callList) Fn;
        Record_Pointer(arrRec) Pns);
external recursive procedure INS9(Record_Pointer(callList) Fn;
        Record_Pointer(arrRec) Pns);
external recursive procedure INS10(Record_Pointer(callList) Fn;
        Record_Pointer(arrRec) Pns);
external recursive procedure INS11(Record_Pointer(callList) Fn;
        Record_Pointer(arrRec) Pns);
external recursive procedure INS12(Record_Pointer(callList) Fn;
        Record_Pointer(arrRec) Pns);
external recursive procedure INS13(Record_Pointer(callList) Fn;
        Record_Pointer(arrRec) Pns);
external recursive procedure INS14(Record_Pointer(callList) Fn;
        Record_Pointer(arrRec) Pns);
external recursive procedure INS15(Record_Pointer(callList) Fn;
        Record_Pointer(arrRec) Pns);

require "PRSNAM.REL[SAM,DGL]" load_module;
external BOOLEAN PROCEDURE ParseName(                    ∂ Written by KS;
     REFERENCE STRING arg;
     REFERENCE STRING device, file);

require "FRMINS.REL[SAM,DGL]" load_module;
external procedure Frmins;



simple procedure ERROR(string errstr);
    begin
    print(errstr,↓,"?");
    inchwl;
    print("Continuing...",↓);
    end;

simple procedure harvy; return;
```

```
∂ Init. break tables;
Integer LDelims, Indel, Crdel;
procedure IniBreaks;
    begin
    ∂ Initialization of break tables;
    breakset(LDelims←getbreak,"<","kins");
    setbreak(indel←getbreak,";"&'12,'15&'14,"insk");
    setbreak(crdel←getbreak,'12,'12&'15&'14&null,"insk");
    TTYUP(true);
    end;
Require IniBreaks initialization;
∂ define mislen="5";
∂ Endgame;
simple procedure ENDGAME;
    begin
            if boxtyp = samdev then PFINISH;
            CLOSE(cmdchan);
            RELEASE(cmdchan);
            close(codchan);
            release(codchan);
            CLOSE(ichan);
            RELEASE(ichan);
    end;
```

```
∂ record handling routines;

procedure ReadinRecordFile(Reference string filestring;
        Reference Record_Pointer(Any_Class)Tops);

∂ this proc. reads in a string of seg function files and makes one
  big linked list out of them, with Tops pointing to the Tops, EndRec
  pointing to the end of the list to facilitate further linking;

while filestring do
     begin
     Record_Pointer(Any_Class) TmpTop;
     Own Record_Pointer(Any_Class) EndRec;

     ParseName(filestring,recdev←"DSK",recfile←"TEST.FUN");
     print("     Reading Func file: ",recdev,":",recfile,↓);
     if Tops=Null_Record
     then
         begin
         fail←segsyn(recdev,recfile,Tops,nfuncs); ∂ read in top record;
         if fail then usererr(0,0,"Can't lookup "&recdev&":"&recfile);
         print(" Record list:");
         EndRec←Tops; ∂ now find the end record;
         while EndRec≠Null_Record do
             begin
             print(" ",seg:name[EndRec]);
             EndRec←seg:next[EndRec];
             end;
         print(↓);
         end
     else
         begin
         fail←segsyn(recdev,recfile,tmpTop,nfuncs); ∂ get Tops record in this file;
         if fail then usererr(0,0,"Can't lookup "&recdev&":"&recfile);
         seg:next[EndRec]←TmpTop; ∂ patch up to EndRec;
         seg:last[TmpTop]←EndRec;
         print(" Record list:");
         EndRec←TmpTop; ∂ now get the new end record;
         while EndRec≠Null_Record do
             begin
             print(" ",seg:name[EndRec]);
             EndRec←seg:next[EndRec];
             end;
         print(↓);
         end
     ;
     end
;

boolean procedure Get_Record(string name;
    reference record_pointer(any_class) r; record_pointer(any_class) Tops);
    begin
    r←Tops;
    while r≠Null_Record do
        begin "LOOP"
        if equ(name,seg:name[r]) then done "LOOP";
        r←seg:next[r];
        end "LOOP";
    if r=Null_Record then return(false) else return(true);
    end;
```

```
procedure LinkRecFun(Record_Pointer(Any_Class) Rf);
    begin
    Own Record_Pointer(CallList) Current;
    if Head=Null_Record
    then
        begin
        Current←Head←New_Record(CallList);
        CallList:TheGoods[Current]←Rf;
        end
    else
        begin
        CallList:Next[Current]←New_Record(CallList); ∂ make new record;
        CallList:Last[CallList:Next[Current]]←Current; ∂ make new record point back;
        Current←CallList:Next[Current]; ∂ move down to new record;
        CallList:TheGoods[Current]←Rf;
        end
    ;
    end;

procedure DelRecList(Reference Record_Pointer(Any_Class) CondemnedTop);
    begin
    Record_Pointer(Any_Class) Foo;
    foo←CondemnedTop;
    while Foo≠Null_Record
    do
        begin
        foo←CallList:next[CondemnedTop];
        $Recfn(5,CondemnedTop);
        CondemnedTop←Foo;
        end
    ;
    end;
```

```
∂ Reserved word and Variable lookup and allocation;

boolean procedure IsDefined(reference string STRN; reference integer type;
        reference real valu; reference integer loc;
        reference Record_Pointer(Seg) FoundRecord);
    begin "SYMBOL_LOOKUP"
        ∂ this proc. is a RAM for defined symbols. It
          takes a string adr, returns a real valu from mem.;
    for loc←0 step 1 until varlen do
        if equ(strn,varids[loc])
        then
            begin
            valu←varlst[loc]; ∂ loc will go back as the adr of this variable;
            type←vardef;
            return(true);
            end
        else
        if ¬varids[loc] ∂ end of declared variables, but not end of space for them;
        then
            done
        ;
    ∂ maybe its an instrument template;
    for loc←0 step 1 while length(tplids[loc]) do
        if equ(strn,tplids[loc])
        then
            begin
            valu←loc; ∂ return addr. of instrument template;
            type←tpldef;
            return(true);
            end
        ;
    ∂ maybe its an instrument;
    for loc←0 step 1 until inslen do
        if equ(strn,insids[loc])
        then
            begin
            valu←inslst[loc]; ∂ return addr. of instrument template;
            type←insdef;
            return(true);
            end
        else
        if ¬insids[loc] ∂ end of declared instruments, but not end of space for them;
        then
            done
        ;
    ∂ if not on variable table, then maybe it's a P field;
    if strn[1 for 1]="P" ∧ "0"≤strn[2 for 1]≤"9"
    then
        begin
        string foo;
        loc←intscan(foo←strn[2 for ∞],brk);
        if loc = 0 then error("Can't address P0, sorry"&↓);
        if loc> prmlen then error("Pn fields only go up to "&cvs(prmlen)&↓);
        valu←prms[loc];
        type←pndef;
        return(true);
        end
    ;
    ∂ maybe it's a record;
    if Get_Record(strn,FoundRecord,Top)
    then
```

```
          begin
          type←recdef;
          valu←0;
          loc←varlen+1;∂ trap to detect an attempt to write into unwritable mem;
          return(true);
          end
     ;
     ∂ maybe it's a symbolic constant;
     for loc←0 step 1 until sclen do
          if equ(strn,scids[loc])
          then
               begin
               type←vardef;
               valu←sclst[loc];loc←varlen+1;
               return(true);
               end
     ;
     ∂ maybe it's a request for new sum memory;
     for loc←0 step 1 until smlen do
          if equ(strn,smids[loc])
          then
               begin
               type←smdef;
               valu←smlst[loc];loc←varlen+1;
               return(true);
               end
     ;
     ∂ maybe it's a request for an OUTn location;
     for loc←0 step 1 until maxchns-1 do
          if equ(strn,outnids[loc])
          then
               begin
               type←vardef;
               valu←outN[loc];
               loc←varlen+1; ∂ kludge to make read-only;
               return(true);
               end
          else
          if equ(strn,outMids[loc])
          then
               begin
               type←vardef;
               valu←outmN[loc];loc←varlen+1;
               return(true);
               end
     ;
     ∂ maybe a packing mode request;
     for loc←0 step 1 until paklen do
     if equ(strn,pakids[loc])
     then
          begin
          type←pakdef;
          valu←paklst[loc];
          loc←varlen+1;
          return(true);
          end
     ;
     ∂ still some things out there that it could be;
     if equ(strn,"ZERO")
     then
          begin
```

```
        type←vardef;
        valu←zero;loc←varlen+1;
        return(true);
        end
    ;
    return(false) ∂ nope, it's not here anywhere, so go home;
    end "SYMBOL_LOOKUP";

integer procedure IsDfntn(string strn);
    if equ("INSTRUMENT",strn) then return(insdef)
    else if equ("RECORD",strn) then return(recdef)
    else if equ("VARIABLE",strn) then return(vardef)
    else return(false)
    ;

procedure STUFSYM(integer type; string name; integer tpladr(0));
    begin "STUF"
    integer nuladr;
    case type of
        begin "FINDEF"
        [insdef]
            for  nuladr←0 step 1 until inslen do
                if equ(insids[nuladr],null)
                then
                    begin
                    insids[nuladr]←name;   ∂ Name of instrument;
                    inslst[nuladr]←tpladr; ∂ This maps to which procedure this name;
                    done;                  ∂ is equated with;
                    end
                else
                if equ(insids[nuladr],name)
                then error("STUFSYM: Mull. Def. Instrument:  "&name)
            ;
        [vardef]
            for  nuladr←0 step 1 until varlen do
                if equ(varids[nuladr],null)
                then
                    begin
                    varids[nuladr]←name;
                    done;
                    end
                else
                if equ(varids[nuladr],name)
                then error("STUFSYM: Mull. Def. Variable:    "&name)
            ;
        else error("STUFSYM: unknown type:    \"&name&"\"&type&↓)
        end "FINDEF"
    ;
    end "STUF";

boolean procedure Contxt(integer current, encountered);
    case encountered of
        begin "CK"
        [pladef] if scnmod=pladef then
            error("Already inside PLAY block!?!! \"&encountered&"\"&↓);
        [findef] if scnmod≠pladef then
            error("FINISH, but no PLAY block!?!!     \"&encountered&"\"&↓);
        [insdef][recdef][vardef] if scnmod=pladef
            ∧ prms[0] ≠ -1 then
            error("Declaration after statement inside PLAY block:    \"&
                encountered&"\"&↓)
        ;
        else
            error("CONTXT: undefined symbol     \"&encountered&"\")
        end "CK";
```

```
∂ INSCAL, figures out whom to call;
procedure INSCAL;
    begin
    ∂ if the parser encounters an instrument call that it must execute,
      then it comes here;

    if boxtyp=samdev
    then
        begin
        Prms[1]←Prms[1]*srate; ∂ convert to samples;
        Prms[2]←Prms[2]*srate;
        if prms[1]>pass then
            begin
            if debug then print
                ("INSCAL doing a Wait_until(time = ",prms[1],") @ curent pass ",
                pass,↓,"          Will then sprout ",insids[prms[0]],prms[1],prms[2],↓);
            wait_until(prms[1]);
            if debug then print
                ("INSCAL waking up @ pass ",pass,↓,"   Will now sprout ",
                insids[prms[0]],prms[1],prms[2],↓);
            end;
        ∂ to get here, all pending wait_untils are after current prms[1];
        if debug then print
            ("INSCAL sprouting @ pass ",pass," ",insids[prms[0]],prms[1],prms[2],↓);
            case prms[0] of
                begin "CALINS"
                [0]     sprout(getitm,ins0(Head,ValueArray(prms)),runme);
                [1]     sprout(getitm,ins1(Head,ValueArray(prms)),runme);
                [2]     sprout(getitm,ins2(Head,ValueArray(prms)),runme);
                [3]     sprout(getitm,ins3(Head,ValueArray(prms)),runme);
                [4]     sprout(getitm,ins4(∂ Head,;ValueArray(prms)),runme);
                [5]     sprout(getitm,ins5(Head,ValueArray(prms)),runme);
                [6]     sprout(getitm,ins6(Head,ValueArray(prms)),runme);
                [7]     sprout(getitm,ins7(Head,ValueArray(prms)),runme);
                [8]     sprout(getitm,ins8(Head,ValueArray(prms)),runme);
                [9]     sprout(getitm,ins9(Head,ValueArray(prms)),runme);
                [10]    sprout(getitm,ins10(Head,ValueArray(prms)),runme);
                [11]    sprout(getitm,ins11(Head,ValueArray(prms)),runme);
                [12]    sprout(getitm,ins12(Head,ValueArray(prms)),runme);
                [13]    sprout(getitm,ins13(Head,ValueArray(prms)),runme);
                [14]    sprout(getitm,ins14(Head,ValueArray(prms)),runme);
                [15]    sprout(getitm,ins15(Head,ValueArray(prms)),runme)
                end "CALINS"
        end
    else
    if boxtyp=frmdev
    then
        Frmins
    else
        error("INSCAL: Unknown boxtyp: "&cvs(boxtyp)&↓)
    ;
    if debug then print
        ("INSCAL returning to parser",↓);
    ∂ exit to parser, which will either lookup a new instrument or see a FINISH;
    end;
```

```
∂ DSK and TTY input routines;

simple string procedure Getstmt;
    begin "GLN" ∂ return line that terminated with semicolon, semicolon and all
    comments removed;
    own string bufstr,bufchr,rtnstr;
    boolean comflg;

    simple string procedure Testex(string filstr);
        begin
        integer i;
        i←0;
        for i←1 step 1 until length(filstr) do
            if filstr[i for 1]="."
            then
                if equ(filstr[i+1 for 3],"FRM")
                then
                    begin
                    boxtyp←frmdev;
                    return("TEST.FRM");
                    end
            ;
        boxtyp←samdev;
        return("TEST.SAM");
        end;

    simple procedure GETFIL;
        begin
        do
            begin
            string tmpnam;
            print(↓,"FILE:    ");
            argument←inchwl;
            if ¬argument
            then
                begin
                readfile←false;
                return;
                end
            else
                begin
                ifile←Testex(argument);
                ParseName(argument,idev←"DSK",ifile);
                open(ichan←getchan,idev,0,2,0,2000,brk,eof);
                lookup(ichan,ifile,fail);
                end
            end
        until ¬fail;
        print(" Reading input file:     ",idev,":",ifile,↓);
        end;

    simple procedure Getlin;
        begin "GIS"
        if ¬readfile
        then
            begin
            print(">");
            bufstr←inchwl;
            end
        else
        if ¬eof
        then
            bufstr←input(ichan,crdel)
        else
        if ichan≠0
        then
            begin
            print("     Closing file:    ",idev,":",ifile,↓);
            close(ichan);
            release(ichan);
            ichan←0;
            end
        else
        if ichan=0 ∂ it is initialized to 0;
        then
            Getfil
        ;
        end "GIS";

    simple procedure Coml;
        if ¬comflg
        then
            rtnstr←rtnstr&bufchr
        ;

    rtnstr←null;
    while true do
        begin "BUFLOOP"
        bufchr←lop(bufstr);
        if bufchr=null
        then
            Getlin
        else
        if bufchr="<"
        then
            Getlin
        else
        if bufchr=";"
        then
            if comflg
            then
                begin
                comflg←false;
                continue "BUFLOOP";
                end
            else
                begin
                if ¬noprint∧readfile then print(rtnstr,↓);
                return(rtnstr);
                end
        else
        if bufchr="C"
        then
            if equ("COMMENT",bufchr&bufstr[1 for 6])
            then
                begin
                comflg←true;
                bufstr←bufstr[7 for ∞]; ∂ sidestep word comment and trailing space;
                end
            else
                Coml
        else
```

```
            Coml
        ;
        end "BUFLOOP";
    end "GLN";
```

```
∂ SCANCALL, the main scan-parse loop;
procedure SCANCALL(string call_string; reference real array prms);
begin "SCN"
        ∂ predicate variables;
        string chr;
        integer LastVarLoc,type;
        real val;
        boolean symlst;


        ∂ stack variables;
        integer opstptr,argstptr;
        own real array argstack[0:64];
        own string array opstack[0:64];

∂ predicate testers;
        boolean procedure ALPHP(string chr);
                if "A"≤chr≤"Z" v chr="_"vchr="$"vchr="#"
                then return(true)
                else return(false);
        boolean procedure SPP(string chr);
                case chr of begin "SPP"
                        ["       "][" "] ["'14"] return(true);
                else return(false)
                end "SPP";
        boolean procedure FMP(string chr);
                if chr =","
                then return(true)
                else return(false);
        boolean procedure NUMP(string chr);
                if "0"≤chr≤"9" then return(true)
                else return(false);
        boolean procedure DOT(string chr);
                if chr="." then return(true) else return(false);
        boolean procedure ATP(string chr);
                if chr="@" then return(true) else return(false);
        boolean procedure OPP(string chr);
                if chr="*"vchr="/"vchr="+"vchr="-"vchr="↑"vchr="←"vchr="("vchr=")"
                then return(true)
                else return(false);

        boolean procedure SAILID(string chr);
                if alphp(chr) v nump(chr) v dot(chr)
                then return(true)
                else return(false);
        boolean procedure COMMA(string chr);
                if chr = "," then return(true)
                else return(false);
        boolean procedure COLONP(string chr);
                if chr = ":" then return(true)
                else return(false);
        boolean procedure NULLP(string chr);
                if chr = null then return(true)
                else return(false);
        boolean procedure PRINTP(string str);
                if equ(str,"PRINT") then return(true)
                else return(false);
```

```
∂ SCANCALL parser;
procedure PUSHARG(real num);
        argstack[argstptr←argstptr+1]←num;

procedure PUSHOP(string opr);
        opstack[opstptr←opstptr+1]←opr;

procedure POPOP(reference string op);
begin "OPPOP"
        op←opstack[opstptr];
        opstptr←opstptr-1;
end "OPPOP";

procedure POPARG(reference real arg);
begin "POPARG"
        arg←argstack[argstptr];
        argstptr←argstptr-1;
end "POPARG";

procedure OPERATE;
    begin "OPRATE"
    real opor,opand;
    string operation;
    if opstack[opstptr] = "⊗" then ∂ "⊗" is symbol here for unary negative;
        begin "UOPNEG"
        argstack[argstptr]←-1*argstack[argstptr];
        if debug then print("¬",argstack[argstptr]);
        opstptr←opstptr-1;
        end "UOPNEG"
    else
    if operation = "⊂"
    then return
    else
        begin "BINOPS"
        popop(operation);
        poparg(opor);
        poparg(opand);
        if operation = "←"
        then
            begin "ASSN"
            if opand > varlen then error("      I think you are trying to"&
                " write into unwritable memory:"&↓&chr&call_string&↓);
            if opand ≥ 0
            then
                begin "ASNVAR"
                if debug then print(varids[opand],operation,opor,↓);
                    ∂ opor has the expression, opand has the index into
                      variable →les;
                varlst[opand]←opor;
                end "ASNVAR"
            else
                begin "ASNPN"
                opand← abs opand; ∂ write into Prms array;
                if debug then print("P",opand,operation,opor,↓);
                    ∂ opor has the expression, opand has the index into
                      variable →les;
                prms[opand]←opor;
                end "ASNPN"
            ;
            pusharg(opor);
            end "ASSN"
        else
            begin "ARITH"
            if debug then print(opand,operation,opor," = ");
            case operation of
                begin "CASE BLOCK"
                ["↑"]  opor←opand↑opor;
                ["*"]  opor←opand*opor;
                ["/"]  opor←opand/opor;
                ["+"]  opor←opand+opor;
                ["-"]  opor←opand-opor
                end "CASE BLOCK";
            pusharg(opor);
            if debug then PRINT(opor,↓);
            end "ARITH"
        ;
        end "BINOPS"
    ;
    end "OPRATE";
```

```
∂ SCANCALL stack manipulation;
boolean procedure JUGGLE(string testop);
begin "JUG"
        simple integer procedure PRECEDENCE(string tst);
                case tst of begin "CASBLK"
                ["⊂"]["("]["←"] return(0);
                ["+"]["-"] return(1);
                ["*"]["/"] return(2);
                ["↑"] return(3);
                ["⊗"] return(4); ∂ stands for unary negative;
                else error("Unrecognized binary op     \"&tst&"\"&↓)
                end "CASBLK";

        boolean procedure PRNOPS(string tesop);
        begin "POPSY"
                if tesop = "("
                then
                    begin
                    pushop(tesop);
                    return(true);
                    end
                else
                if tesop = ")"
                then
                    begin
                    while opstack[opstptr]≠"(" do
                        begin
                        OPERATE;
                        if opstack[opstptr]="⊂" then
                                error("Parenthesis underflow"&↓);
                        end;
                    opstptr←opstptr-1; ∂ dump the left paren;  <———— uh...
                    return(true);
                    end
                else return(false);
        end "POPSY";

        boolean procedure ENDOP(string testop);
            begin "ENDS"
            if testop = "⊃" then
                begin
                while opstack[opstptr]≠"⊂" do
                    begin
                        OPERATE;
                        if opstack[opstptr]="(" then
                            error("Parenthesis overflow"&↓);
                    end;
                return(true);
                end
            else return(false);
            end "ENDS";

        boolean procedure ASSNOP(string testop);
                ∂ the assignment is presumably to a variable whose value has
                  just been pushed on argstack, so pop the stack, and push
                  instead its location in varlst, which we've cleverly
                  kept in LastVarLoc just for this purpose;
            if testop = "←" then
                begin
                argstack[argstptr]←LastVarLoc;
                return(true);
```

```
                        end
                    else return(false);

                if ¬ENDOP(testop)
                then
                    begin
                    if ¬PRNOPS(testop)
                    then
                        if ¬ASSNOP(testop)
                        then
                            if precedence(testop) - precedence(opstack[opstptr]) ≤ 0
                            ∂ ie if precedence of current op ≤ that of previous op;
                            then OPERATE
                    ;
                    pushop(testop);
                    end
                ;
end "JUG";
```

```
∂ SCANCALL scan routines;
procedure ENDOFIELD;
     begin "EOF"
     JUGGLE("⊃");
     if fldmod
     then
         begin
         fldctr←fldctr+1;
         if boxtyp=frmdev
         then
             if tmpins≠-1 ∧ fldctr=1
             then
                 begin
                 poparg(TmpP1);
                 INSCAL;
                 prms[0]←tmpins; ∂ replace with current instrument adr;
                 prms[fldctr]←TmpP1; ∂ put top of argstack in prms[1];
                 end
             else poparg(prms[fldctr])
         else
         if boxtyp=samdev
         then
             poparg(prms[fldctr]) ∂ put val. in prms;
         ;
         end
     else argstptr←argstptr-1 ∂ throw it away;
     ;
     symlst←false;
     end "EOF";

string procedure GETSTR;
begin "ALPNOD"
        string object;
        object←null;
        while alphp(chr)∨nump(chr) do
        begin "DKD"
                object←object&chr;
                chr←lop(call_string);
        end "DKD";
        return(object);
end "ALPNOD";

real procedure GETNUM;
begin "NUMNOD"
        integer brk;
        real v;
        call_string←chr&call_string;
        v←realscan(call_string,brk);
        chr←lop(call_string); ∂ rlscn uses retain mode, so must do a lop
                to prime chr with next character;
        symlst←true;
        return (v);
end "NUMNOD";

procedure GETBIN(string tstop);
begin "BINNOD"
        JUGGLE(tstop);
        if tstop=")" then symlst←true else symlst←false;
end "BINNOD";

procedure GETUOP(string tststr);
```

```
        case tststr of begin "GUOP"
        ["-"] pushop("⊗");
        [")"] JUGGLE(tststr);
        ["("]   pushop(tststr);
        else if tststr≠"←" then
                error("Undefined unary op       \"&tststr&"\"&↓)
        end "GUOP";
```

```
∂ SCANCALL reserved word scan procedures;

procedure DefinIt(integer kind);
    begin "SYMMAK"
    if kind = recdef
    then
        begin
        ReadinRecordFile(call_string,Top);
        chr←null;
        end
    else
    if kind = insdef
    then
        begin ∂ <template> <field mark> <name> <field mark> <name> ...;
        while spp(chr) v fmp(chr) do chr←lop(call_string); ∂ strip spaces;
        if alphp(chr) ∂ will be first chr in template name;
        then
            begin
            string tplstr;
            integer T,LVL,R;
            real V;
            record_pointer (any_class)Rfoo;
            tplstr←GETSTR;
            if IsDefined(tplstr,T,V,LVL,Rfoo)
            then ∂ IsDefined won, so let's see what it got;
                if T = tpldef
                then do ∂ ok, we have a valid template type, addr is in V;
                    begin ∂ now go suck up names to be associated with it;
                    while spp(chr) v fmp(chr) do chr←lop(call_string); ∂ strip spaces;
                    if alphp(chr) ∂ will be first chr in template name;
                    then
                        begin
                        string inst;
                        inst←GETSTR;
                        StufSym(kind,inst,V);
                        end
                    else
                        if chr ≠ null then error("DEFINIT: Symbols must begin with alphanum
eric"&
                            "       \"&chr&call_string&"\"&↓);
                    end until ¬chr
                else
                    error("DEFINIT: Illegal instrument template name: \"&tplstr&"\"&↓)
            else ∂ IsDefined lost;
                error("DEFINIT: Undefined template: \"&tplstr&"\"&↓);
            end
        else
            if chr ≠ null then error("DEFINIT: Symbols must begin with alphanumeric"&
                "       \"&chr&call_string&"\"&↓);
        end
    else
    if kind = vardef
    then do  ∂ snarf the variable names;
        begin
        while spp(chr) v fmp(chr) do chr←lop(call_string); ∂ strip spaces;
        if alphp(chr) ∂ will be first chr in name;
        then
            begin
            string tmpstr;
            tmpstr←GETSTR;
            STUFSYM(kind,tmpstr);
```

```
                end
            else
                if chr ≠ null then error("DEFINIT: Symbols must begin with alphanumeric"&
                    "        \"&chr&call_string&"\"&↓);
            end until ¬chr
        ;
        end "SYMMAK";
```

```
∂ SCANCALL reserved word scan procedures continued;

simple procedure SETPLAY(reference string xchr,xcall_string);
    begin
    while xchr≠null ∧ ¬sailid(xchr) do xchr←lop(xcall_string); ∂ get to the filename;
    argument←xchr&xcall_string;
    xcall_string←xchr←null;
    ParseName(argument,cmddev←"DSK",cmdfile←"TEST.DOA");
    print("     Command output to file:         ",cmddev,":",cmdfile,↓);
    if boxtyp=false
    then
        begin
        print(" Default output to Samson box.",↓);
        boxtyp←samdev;
        end
    ;
    if boxtyp=samdev
    then
        begin
        open(cmdchan←getchan,cmddev,'17,0,0,0,brk,eof);
        enter(cmdchan,cmdfile,fail);
        open(codchan←getchan,coddev←cmddev,'17,0,0,0,brk,eof);
        enter(codchan,codfile←"test.cod",fail);
        ∂ init box code;
        Set_Output(Code_Stream,CmdChan);
        Set_Output(Command_Stream,CodChan);
        set_mode(non_optimize);
        if ¬NPTIX v ¬NUTIX
        then
            begin
            print("     Default nProc. Tix. = 96, nUpd. Tix. = 32 set",↓);
            NPTIX←96;
            NUTIX←32;
            end
        ;
        if ¬NOUTCHANS
        then
            begin
            print("     Default Nchans = 4, on generator side of sum memory",↓);
            NOUTCHANS←4;
            end
        ;
        Pinit(NOUTCHANS,WHICHSIDE,FILTERS,NPTIX,NUTIX);
        if ¬pakSet
        then
            begin
            set_field(packing_mode,full_word);
            pak←full_word;
            print(" Default packing mode = full_word",↓);
            end
        ;
        end
    else
    if boxtyp=frmdev
    then
        begin
        open(cmdchan←getchan,cmddev,0,0,2,0,brk,eof);
        enter(cmdchan,cmdfile,fail);
        cprint(cmdchan,".0ve←←<1000,,0>",↓);
        end
    ;
    if fail then usererr(0,0,"Can't enter "&cmddev&":"&cmdfile);
    end;


boolean procedure SetBlk(string blkstr; reference string zchr, zcallstring);
    begin "BKST"
    integer tstmod;
    if equ("PLAY",blkstr)          then tstmod←pladef
    else if equ("FINISH",blkstr)   then tstmod←findef
    else if equ("BEGIN",blkstr)    then tstmod←begdef
    else if equ("END",blkstr)      then tstmod←enddef
    else return(false)
    ;
    if CONTXT(scnmod,tstmod) then scnmod←tstmod;
    if scnmod=findef ∧ prms[0]≠-1
    then ∂ flush last instrument call;
        begin
        if boxtyp=samdev
        then
            begin
            INSCAL;
harvy;
            ∂ now force monitor to run last insts.;
            if debug then print("FINISH about to wait_until ",prms[1],"+",prms[2],
                "+1 =",prms[1]+prms[2]+1,"@ pass = ",pass,↓);
            wait_until(prms[1]+prms[2]+1); ∂ 1 greater than last instrument!;
            pass←0;
harvy;
            PFINISH;
harvy;
            end
        else
        if boxtyp=frmdev
        then
            begin
            tmpP1←prms[1]+1; ∂ a hack to flush the last instrument call;
            Inscal;
            end
        ;
        tmpins←prms[0]←-1; ∂ Reset null instrument;
        close(cmdchan);
        release(cmdchan);
        close(codchan);
        release(codchan);
        codchan←cmdchan←0;
        print(" Finishing command output file   ",cmddev,":",cmdfile,↓);
        if boxtyp=frmdev
        then
            print("                                 and ",coddev,":",codfile,↓)
        ;
        end
    else
    if scnmod=pladef
    then
       SETPLAY(zchr,zcallstring);
    ;
    return(true);
    end "BKST";
```

```
∂ SCANCALL main scan loop;
symlst←false;
pushop("c");
LastVarLoc←-1; ∂ init. to mean it's not pointing at a variable location;

chr←lop(call_string); ∂ seed chr;

while chr do
begin "SCAN_LOOP"
string strid;
while spp(chr) do chr←lop(call_string); ∂ strip spaces, tabs, etc;
if sailid(chr)
then ∂ +sailid;
    if symlst
    then ∂ +symlst.  Symlst says there was a symbol last with no intervening binop;
        ENDOFIELD  ∂ don't take next chr;
    else ∂ ¬symlst, so we can now look at new symbol;
    if nump(chr) v dot(chr)
    then ∂ +number;                          [illegible]
        begin "ISNUM"
        val←GETNUM; ———————— [illegible]
        pusharg(val);
        end "ISNUM"
    else
    if alphp(chr)
    then ∂ +alphp;
        begin "ISALPH" ∂ can be any of a variable, reserved word or inst. name;
        strid←GETSTR;       ∂ soak up symbol;
        if IsDefined(strid,type,val,LastVarLoc,Rec)
        then ∂ means it is defined;
            if type = vardef
            then
                begin "ISV"
                symlst←true; ∂ we just snarfed a variable;
                pusharg(val); ∂ value of variable is in val;
                end "ISV"
            else
            if type = pndef
            then
                begin "ISPN"
                symlst←true;
                pusharg(val);
                LastVarLoc←-LastVarLoc; ∂ hack to force lookup of Pn, not Variable;
                end "ISPN"
            else
            if type = recdef
            then
                begin
                symlst←true;
                pusharg(val); ∂ stack must be pushed, but doesn't do anything;
                if NewInstStarted
                then
                    begin
                    DelRecList(Head); ∂ scratch old record call list;
                    NewInstStarted←false;
                    end;
                LinkRecFun(Rec);
                end
            else
            if type = smdef
            then
```

```
                begin
                pusharg(getsm(val));
                if argstack[argstptr]=true
                then
                    error("Ran out of sum memory!"&↓);
                end
            else
            if type = insdef ∂ it is a call to an instrument;
            then
                begin
                if scnmod≠pladef
                then
                    error("Call to instrument outside PLAY block!       "&strid&↓)
                ;
                if prms[0]≠-1 ∂ i.e., if the first instrument has been called;
                then
                    if boxtyp=samdev
                    then
                        begin
                        INSCAL;
                        prms[fldctr←0]←val; ∂ addr. of instrument is always in P0;
                        end
                    else
                    if boxtyp=frmdev
                    then
                        begin
                        tmpins←val;
                        fldctr←0;
                        ∂ now continue until P1 is parsed, then do INSCAL;
                        end
                    else error("SCANLOOP: boxtyp error"&↓)
                else
                    begin
                    NewInstStarted←true;
                    prms[fldctr←0]←val;
                    end
                ;
                fldmod←true; ∂ initialize field scanning mode;
                end
            else
            if type = tpldef
            then
                error("Template name illegal here, use only in instrument "&
                    "definitions, please")
            else error("Undefined type =        "&cvs(type))
        else
        if Type←IsDfntn(strid) ∂ any of INSTRUMENT, RECORD, PLAY, FINISH, etc.;
        then
            DefinIt(type) ∂ suck up the arguments;
        else
        if Printp(strid)
        then ∂ flush print statements;
            begin
            chr←call_string←null;
            done "SCAN_LOOP";
            end
        else
        if type = pakdef
        then
            begin
            set_field(packing_mode,val);
```

```
                pakSet←true;
                end
            else
            if ¬SetBlk(strid,chr,call_string)
            then
                error("Unknown symbol       \"&strid&"\"&↓)
            ;
            end "ISALPH"
        else error("Unknown condition = "&chr&↓)
    else ∂ ¬sailid;
    if opp(chr)
    then ∂ +opp;
        begin "ISOP"
        if symlst
        then ∂ +symlst;
            if chr="("
            then ∂ +chr="(";
                begin "UnaryLeftPrn"
                ENDOFIELD;
                continue "SCAN_LOOP"; ∂ don't lop next chr;
                end "UnaryLeftPrn"
            else ∂ +binary op;
                GETBIN(chr)
        else ∂ symbol not read last time, so is unary op;
            GETUOP(chr)
        ;
        chr←lop(call_string);
        end "ISOP"
    else ∂ ¬sailid ∧ ¬op, so maybe a comma?;
    if fmp(chr)
    then
        begin "ISCOM"
        if argstptr<1 then pusharg(prms[fldctr+1]);
        ENDOFIELD;
        chr←lop(call_string);
        end "ISCOM"
    else
    if nullp(chr)
    then
        done "SCAN_LOOP"
    else ∂ I give up!;
        begin
        integer foo;
        foo←cvo(foo);
        error("Don't know what to do with "&cvs(foo)&" = \"&chr&"\"&↓);
        end
    ;
    end "SCAN_LOOP";
    if argstptr<1 then pusharg(0);
    ENDOFIELD;
    opstptr←opstptr-1;
    fldmod←false;
    end "SCN";
```

?

```
ð main prog;
    debug←noprint←false;
    fldmod←false;
    eof←readfile←fnin←true;
    tmpins←prms[0]←-1;
    while true do
        begin "FUB"
        stmt←Getstmt;
        if equ(stmt,"EXIT")
        then
            done"FUB"
        else
        if equ(stmt,"FILE")
        then
            fnin←readfile←true
        else
            SCANCALL(stmt, prms)
        ;
        end "FUB";
ENDGAME;
end "MBOX";
```