Name: Gareth Loy

Project: SAM Programmer: DGL

File Name: SAMINS.SAI[SAM,DGL]

File Last Written: 11:21 3 Jan 1978

Time: 11:27 Date: 3 Jan 1978

Stanford University
Artificial Intelligence Laboratory
Computer Science Department
Stanford, California

```
COMMENT ⊗   VALID 00017 PAGES
C REC  PAGE   DESCRIPTION
C00001 00001
C00003 00002    ENTRY "INSTS"
C00004 00003    ∂ GOSC - fixed frequency, seg record variable amplitude
C00008 00004    ∂ GOSCVF - seg record variable frequency and amplitude
C00012 00005    ∂ SHAPE - apply an envelope to a signal of amp up to 8.0
C00015 00006    ∂ LINGER - issues a dwell command from times p1 to p2, facility to clear
uses
C00017 00007    ∂ GEN - raw call to a generator, with scaling of frq and sum_of_cosines
C00023 00008    ∂ MOD - raw call to a modifier, no scaling of any terms
C00026 00009    ∂ MIX - Multiply two signals by a constant factors between 0 and 8.
C00028 00010    ∂ RANDOM - Make random numbers scaled by a factor
C00030 00011    ∂ FILTER - Does one or two pole or zero fixed character filtering.
C00032 00012    ∂ REV - Reverberator. Can be all-pass or comb.
C00034 00013    ∂ GENFM1
C00039 00014    ∂ GENFM3
C00045 00015    ∂ NGEN
C00048 00016    Internal recursive procedure insl3(Record_Pointer(rpRec) rPns
C00049 00017    END "INSTS"
C00050 ENDMK
C⊗;
```

```
ENTRY "INSTS";
BEGIN "INSTS";

require "MBOX.HDR[SAM,DGL]" source_file;
require "SAMTBL.SAI[SAM,DGL]" source_file;
external procedure BoxError(string errmsg);
```

```
∂ GOSC - fixed frequency, seg record variable amplitude;

∂ sample call:
    <instrument> Beg, Dur, Freq, ampRec, Ampscl, Ampoff, Mod, Ncs, Fmsum, outsum;

Internal recursive procedure ins0(Record_Pointer(rpRec) rPns;
        Record_Pointer(ArrRec) Pns;integer typOut);
    begin
    itemvar lsegItem;
    integer gen;
    real array locPns[0:10]; ∂ local array to have parameters transferred into;
    record_pointer(seg) locRec; ∂ local pointer to the function
                                record for this inst;

    arrtran(locPns,ArrRec:X[Pns]); ∂ transfer from global Pns to local Pns;
    $RECFN(5,Pns); ∂ delete global Pns array record pointer;

    locRec←rpRec:rpArr[rPns][4]; ∂ locRec now points to function record;
    $RECFN(5,rPns); ∂ delete global rPns array record pointer;

∂   Internal integer procedure GOSC(integer stsamp,nsamps| real freq|
            record_pointer(seg) ampRec|          ! This is the amplitude function|
            real ampscl,ampoff|                  ! Scale factor and offset for amplitude|
            reference itemvar fitem|             ! Returns amp rPns process item here|
            integer mod,ncs,                     ! Generator mode, number of cosines|
                fmsum,outsum                     ! FM input, output location|);

    if typOut land debug_instruments
    then
        PRINT("GOSC: ", CallNo, myproc, locPns[1]/srate, locPns[2]/srate,↓,
            tab, locPns[1], locPns[2], locPns[3], seg:name[locRec], locPns[5], ↓,
            tab, locPns[6], locPns[7], locPns[8], locPns[9], locPns[10],↓)
    ;

    GEN←GOSC(locPns[1],locPns[2],locPns[3],
        locRec, ∂ locPns[4] will be zero, since that's where locRec goes;
        locPns[5],locPns[6],
        lsegItem,
        locPns[7],locPns[8],
        locPns[9],locPns[10])
    ;
    Join({lsegItem});
    give(gen);
    if typOut land debug_instruments
    then
        print("    GOSC: ",myproc, locPns[1]/srate, locPns[2]/srate,
         " Dropping off end",↓);
    rtnitm(myproc);
    end;
```

```
∂ GOSCVF - seg record variable frequency and amplitude;

∂ Sample call in an MBOX play list:
    <instrument>, beg, dur, frqRec, Frscl, Froff, ampRec, Ampscl, Ampoff,
    Mod, Ncs, Fmsum, Outsum;

    COMMENT here are the formals for the call to INTERM;
    ∂ integer procedure GOSCVF(integer stsamp,nsamps|
        record_pointer(seg) frfn|       ! This is the frequency function|
        real frscl,froff|               ! Scale factor and offset for frequency|
        reference itemvar fritem|       ! Returns frq rPns process item here|
        record_pointer(seg) rPns|       ! This is the amplitude function|
        real ampscl,ampoff|             ! Scale factor and offset for amplitude|
        reference itemvar fitem|        ! Returns amp rPns process item here|
        integer mod,ncs,                ! Generator mode, number of cosines|
            fmsum,outsum                ! FM input, output location|);

Internal recursive procedure insl(Record_Pointer(rpRec) rPns;
        Record_Pointer(ArrRec) Pns; integer typOut);
    begin
    Record_Pointer(seg)frqRec, ampRec;
    integer gen;
    itemvar ampitm,frqitm;
    real array locPns[0:12]; ∂ local array to have parameters transferred into;

    arrtran(locPns,ArrRec:X[Pns]); ∂ transfer from global Pns to local Pns;
    $RECFN(5,Pns); ∂ delete global Pns array record pointer;

    frqRec←rpRec:rpArr[rPns][3]; ∂ this one for frequency;
    ampRec←rpRec:rpArr[rPns][6]; ∂ this one for amplitude;
    $RECFN(5,rPns); ∂ delete global rPns array record pointer;

    if typOut land debug_instruments
    then
        PRINT("GOSCVF: ", CallNo, myproc, locPns[1]/srate, locPns[2]/srate,↓,
            tab, locPns[1], locPns[2], seg:name[frqRec], locPns[4], locpns[5],↓,
            tab, seg:name[ampRec], locPns[7], locPns[8], locPns[9],↓,
            tab, locPns[10],locPns[11],locPns[12],↓);
    ;
    Gen←Goscvf(locPns[1],locPns[2],
        frqRec,          ∂ locPns[3] unused;
        locPns[4],locPns[5],
        frqItm,
        ampRec,          ∂ locPns[6] unused;
        locPns[7],locPns[8],
        ampItm,
        locPns[9],locPns[10],
        locPns[11],locPns[12]);

    join({frqItm,ampItm});
    give(gen);
    if typOut land debug_instruments then
        print(" GOSCVF: ",myproc, locPns[1]/srate, locPns[2]/srate,
            " Dropping off end",↓);
    rtnitm(myproc);
    end;
```

```
∂ SHAPE - apply an envelope to a signal of amp up to 8.0;

∂ Sample call:
    <instrument> beg,dur,ampRec,scale,offset,outadr,inadr,factor;

Internal recursive procedure ins2(Record_Pointer(rpRec) rPns;
        Record_Pointer(ArrRec) Pns; integer typOut);
    begin
    Record_Pointer(seg)frqRec, ampRec;
    integer gen,mpymod,sum;
    itemvar ampItm;
    real array locPns[0:8]; ∂ local array to have parameters transferred into;
    arrtran(locPns,ArrRec:X[Pns]); ∂ transfer from global Pns to local Pns;
    $RECFN(5,Pns); ∂ delete global Pns array record pointer;

    ∂ Get the first record on the rPns of records sent to this instrument;
    ampRec←rpRec:rpArr[rPns][4];
    $RECFN(5,rPns); ∂ delete global rPns array record pointer;

    ∂ Internal integer procedure SHAPE(integer stsamp,nsamps|
            record_pointer(seg) rPns|          ! This is the amplitude function|
            real ampscl,ampoff|              ! Scale factor and offset for amplitude|
            reference itemvar fitem|         ! Returns amp rPns process item here|
            integer outloc,inloc|            ! Sum memory locs of output and input|
            reference integer mod,sum|       ! Place to put modifier and sum mem|
            real factor                      ! Multiply whole thing by this|);

    if typOut land debug_instruments
    then
        PRINT("SHAPE: ", CallNo, myproc, locPns[1]/srate, locPns[2]/srate,↓,
            tab, locPns[1], locPns[2], seg:name[ampRec], locPns[4], ↓,
            tab, locPns[5], locPns[6], locPns[7], locPns[8], ↓)
    ;

    gen←Shape(locPns[1],locPns[2],
        ampRec,
        locPns[4],locPns[5],
        ampItm,
        locPns[6],locPns[7],
        mpymod,sum,
        locPns[8]);

    join({ampItm});
    give(gen);
    give(sum);
    give(mpymod);
    if typOut land debug_instruments then
        print("    SHAPE: ",myproc, locPns[1]/srate, locPns[2]/srate,
            " Dropping off end",↓);
    rtnitm(myproc);
    end;
```

```
∂ LINGER - issues a dwell command from times p1 to p2, facility to clear pauses;

∂ sample call:
    <instrument> beg dur;

Internal recursive procedure ins3(
        ∂ Record_Pointer(rpRec) rPns;
        Record_Pointer(ArrRec) Pns; integer typOut);
    begin
    real array locPns[0:3]; ∂ local array to have parameters transferred into;
    arrtran(locPns,ArrRec:X[Pns]); ∂ transfer from global Pns to local Pns;
    $RECFN(5,Pns); ∂ delete global Pns array record pointer;

    if typOut land debug_instruments
    then
        PRINT("LINGER: ",CallNo," ",myproc, locPns[1]/srate, locPns[2]/srate,
            locPns[3],↓);

    wait_until(locPns[1]);
    if locPns[2]>0
    then
    set_field(dwell,locPns[2]);
    if locPns[3]>0
    then
        set_mode(locPns[3]);

    wait_until(locPns[1]+locPns[2]);
    if typOut land debug_instruments then
        print(tab,"LINGER: ",myproc, locPns[1]/srate, locPns[2]/srate,
            " Dropping off end",↓);
    rtnitm(myproc);
    end;
```

```
∂ GEN - raw call to a generator, with scaling of frq and sum_of_cosines;

∂ sample call:
    <instrument> Beg, Dur, Freq, Sweep, Angle, nCos, Scale,
    Asym, Rate, Exp, Mod, Fmsum, Outsum, GenNum;
∂ Some things, but not many, are done for you here:
        1. Frq is scaled by mag.
        2. If Scale is 0, calculate Scale from nCos.
        3. If GenNum>0 then claim that generator, else take first available.
;

Internal recursive procedure ins4(
        ∂ Record_Pointer(rpRec) rPns;
        Record_Pointer(ArrRec) Pns; integer typOut);
    begin
    integer gen;
    real array locPns[0:14]; ∂ local array to have parameters transferred into;
    arrtran(locPns,ArrRec:X[Pns]); ∂ transfer from global Pns to local Pns;
    $RECFN(5,Pns); ∂ delete global Pns array record pointer;

    if locPns[14]>0 then
        gen←get(id_generator,locPns[14])
    else
        gen←get(id_generator)
    ;
    if gen < 0 then BoxError("GEN: Can't claim that generator!"&
        tab&"gen: "&cvs(gen)&" P14: "&cvs(locPns[14]));

    if typOut land debug_instruments then
        PRINT("GEN: ",CallNo," ",myproc," ",gen, locPns[1]/srate, locPns[2]/srate, ↓,
        tab,locPns[1], locPns[2], locPns[3], locPns[4], locPns[5], ↓,
        tab,locPns[6], locPns[7], locPns[8], locPns[9], locPns[10], ↓, tab,
        locPns[11], locPns[12], locPns[13], locPns[14],↓);

    bind(gen,mode,g_inactive);
    bind(gen,sum_memory,locPns[13]); ∂ outsum;
    bind(gen,sweep,locPns[4]);
    bind(gen,frequency,locPns[3]*mag);
    bind(gen,fm,locPns[12]);
    bind(gen,asymptote,locPns[8]);
    bind(gen,exponent,locPns[10]);
    bind(gen,rate,locPns[9]);
    bind(gen,angle,locPns[5] ∂ '4000000*locPns[5]/(2*pi););
    bind(gen,scale,locPns[7]);
    bind(gen,ncosines,locPns[6]);

    wait_until(locPns[1]);
    bind(gen,mode,locPns[11]);
    wait_until(locPns[1]+locPns[2]);
    bind(gen,rate,0);
    bind(gen,sweep,0);
    bind(gen,mode,g_inactive);
    if typOut land debug_instruments then
        print("    GEN: ",myproc, locPns[1]/srate, locPns[2]/srate,
            " Dropping off end",↓);
    rtnitm(myproc);
    end;
```

```
∂ MOD - raw call to a modifier, no scaling of any terms;

∂ sample call:
    <instrument> beg dur cf0 cf1 trm0 trm1 ain bin outloc ascl bscl mode modnum;
    ∂ If modnum>0 then claim that modifier, else take the first that's available;

Internal recursive procedure ins5(
        ∂ Record_Pointer(rpRec) rPns;
        Record_Pointer(ArrRec) Pns; integer typOut);
    begin
    integer mod;
    real array locPns[0:13]; ∂ local array to have parameters transferred into;
    arrtran(locPns,ArrRec:X[Pns]); ∂ transfer from global Pns to local Pns;
    $RECFN(5,Pns); ∂ delete global Pns array record pointer;

    if typOut land debug_instruments then
    PRINT("MOD: ",CallNo," ", myproc, locPns[1]/srate, locPns[2]/srate, ↓,
        locPns[1], locPns[2], locPns[3], locPns[4],↓,
        locPns[5], locPns[6], locPns[7], locPns[8], locPns[9],
        locPns[10], locPns[11], locPns[12], locPns[13],↓);

    if locPns[13]>0 then
        mod←get(id_modifier,locPns[13])
    else
        mod←get(id_modifier);
    bind(mod,mode,m_inactive);

    bind(mod,coeff0,locPns[3]);
    bind(mod,coeff1,locPns[4]);
    bind(mod,term_0,locPns[5]);
    bind(mod,term_1,locPns[6]);
    bind(mod,A_in,locPns[7]);
    bind(mod,B_in,locPns[8]);
    bind(mod,sum_memory,locPns[9]);
    bind(mod,A_scale,locPns[10]);
    bind(mod,B_scale,locPns[11]);

    wait_until(locPns[1]);
    bind(mod,function,locPns[12]);
    wait_until(locpns[1]+locPns[2]);
    bind(mod,mode,m_inactive);
    if typOut land debug_instruments then
        print("    MOD: ",myproc, locPns[1], locPns[2], " Dropping off end",↓);
    rtnitm(myproc);
    end;
```

```
∂ MIX - Multiply two signals by a constant factors between 0 and 8.

∂ sample call:
    <instrument> beg dur factor1 factor2 in1loc in2loc outloc;

∂ MIX(integer outloc,in1loc| real factor1|
        integer in2loc| real factor2);

Internal recursive procedure ins6(
        ∂ Record_Pointer(rpRec) rPns;
        Record_Pointer(ArrRec) Pns; integer typOut);
    begin
    ∂ Record_Pointer(seg)ampRec;
    integer mod;
    real array locPns[0:7]; ∂ local array to have parameters transferred into;
    arrtran(locPns,ArrRec:X[Pns]); ∂ transfer from global Pns to local Pns;
    $RECFN(5,Pns); ∂ delete global Pns array record pointer;

    if typOut land debug_instruments then
    PRINT("MIX: ",CallNo," ", myproc, locPns[1]/srate, locPns[2]/srate, ↓,
        locPns[1],locPns[2],locPns[3],locPns[4],locPns[5],↓,
        locPns[6], locPns[7],↓);

    wait_until(locpns[1]);
    mod ← MIX(locPns[7],locPns[5],locPns[3],
        locPns[6],locPns[4]);

    wait_until(locpns[1]+locpns[2]);
    if typOut land debug_instruments then
        print("    MIX: ",myproc, locPns[1], locPns[2], " Dropping off end",↓);
    give(mod);
    rtnitm(myproc);
    end;
```

```
∂ RANDOM - Make random numbers scaled by a factor;

∂ sample call:
    <instrument> beg dur factor trigger seed outloc;

∂ RANDOM(integer outloc| real factor|
        reference integer mod1,mod2,sum| integer trigger(0),seed(0));

Internal recursive procedure ins7(
        ∂ Record_Pointer(rpRec) rPns;
        Record_Pointer(ArrRec) Pns; integer typOut);
    begin
    ∂ Record_Pointer(seg)ampRec;
    integer mod1, mod2, sum;
    real array locPns[0:6]; ∂ local array to have parameters transferred into;
    arrtran(locPns,ArrRec:X[Pns]); ∂ transfer from global Pns to local Pns;
    $RECFN(5,Pns); ∂ delete global Pns array record pointer;

    if typOut land debug_instruments then
    PRINT("RANDOM: ",CallNo," ", myproc, locPns[1]/srate, locPns[2]/srate, ↓,
        locPns[1],locPns[2],locPns[3],locPns[4],locPns[5],
        locPns[6],↓);

    wait_until(locPns[1]);
    RANDOM(locPns[6], locPns[3],
        mod1, mod2, sum, locPns[4], locPns[5]);

    wait_until(locPns[1]+locPns[2]);
    if typOut land debug_instruments then
        print("    RANDOM: ",myproc, locPns[1], locPns[2], " Dropping off end",↓);
    give(mod1);
    give(mod2);
    give(sum);
    rtnitm(myproc);
    end;
```

```
∂ FILTER - Does one or two pole or zero fixed character filtering.

∂ sample call:
    <instrument> beg dur frq R second_order all_pole inloc outloc;

Internal recursive procedure ins8(
        ∂ Record_Pointer(rpRec) rPns;
        Record_Pointer(ArrRec) Pns; integer typOut);
    begin
    ∂ Record_Pointer(seg)ampRec;
    integer mod;
    real array locPns[0:8]; ∂ local array to have parameters transferred into;
    arrtran(locPns,ArrRec:X[Pns]); ∂ transfer from global Pns to local Pns;
    $RECFN(5,Pns); ∂ delete global Pns array record pointer;

    if typOut land debug_instruments then
    PRINT("FILTER: ",CallNo," ", myproc, locPns[1]/srate, locPns[2]/srate, ↓,
        locPns[1],locPns[2],locPns[3],locPns[4],locPns[5],↓,
        locPns[6],locPns[7],locPns[8],↓);

    ∂ FILTER(integer outloc,inloc| real R,freq|
            boolean second_order(true),all_pole(true));

    wait_until(locpns[1]);
    mod ← FILTER(locPns[8], locPns[7], locPns[4], locPns[3],
        locPns[5], locPns[6]);

    wait_until(locpns[1]+locpns[2]);
    if typOut land debug_instruments then
        print("    FILTER: ",myproc, locPns[1], locPns[2], " Dropping off end",↓);
    give(mod);
    rtnitm(myproc);
    end;
```

```
∂ REV - Reverberator. Can be all-pass or comb.
To make an allpass, set g1←-g0←G;

∂ sample call:
    <instrument> beg dur length g1 g2 inloc outloc;

Internal recursive procedure ins9(
        ∂ Record_Pointer(rpRec) rPns;
        Record_Pointer(ArrRec) Pns; integer typOut);
    begin
    integer mod, dly, dlyAdr;
    real array locPns[0:7]; ∂ local array to have parameters transferred into;
    arrtran(locPns,ArrRec:X[Pns]); ∂ transfer from global Pns to local Pns;
    $RECFN(5,Pns); ∂ delete global Pns array record pointer;

    if typOut land debug_instruments then
    PRINT("REV: ",CallNo, " ",myproc, locPns[1]/srate, locPns[2]/srate, ↓,
        locPns[1],locPns[2],locPns[3],locPns[4],locPns[5],↓,
        locPns[6],locPns[7],↓);

    wait_until(locpns[1]);      ∂ Get to start of note;

    ∂ REV(integer outloc,inloc|
            real g0,g1| integer length|
            reference integer dly,dlyAdr);

    mod ← REV(locPns[7], locPns[6],
        locPns[4], locPns[5], locPns[3],
        dly, dlyAdr);

    wait_until(locpns[1]+locpns[2]);
    if typOut land debug_instruments then
        print("    REV: ",myproc, locPns[1], locPns[2], " Dropping off end",↓);
    give(mod);
    give(dly);
    dmrel(dlyAdr);
    rtnitm(myproc);
    end;
```

```
∂ GENFM1;

∂ sample call:
    <instrument> Beg, Dur, Freq, ampRec, Ampscl, Ampoff,
    Mode, Ncs, cfmfRatio, indexRec, index, indexOff, outsum;

Internal recursive procedure ins10(Record_Pointer(rpRec) rPns;
        Record_Pointer(ArrRec) Pns; integer typOut);
    begin
    itemvar FlsegItem, ClsegItem;
    integer genCf, genMf, sumMf;
    real peakDev, normDev, maxDev, mFq, cFq, normIndOff,indx;

    real array locPns[0:13]; ∂ local array to have parameters transferred into;
    record_pointer(seg) locRec, locFm; ∂ local pointer to the function
                                record for this inst;

    arrtran(locPns,ArrRec:X[Pns]); ∂ transfer from global Pns to local Pns;
    $RECFN(5,Pns); ∂ delete global Pns array record pointer;

    locRec←rpRec:rpArr[rPns][4]; ∂ locRec now points to function record;
    locFm←rpRec:rpArr[rPns][10]; ∂ locFm is for the fm index function;
    $RECFN(5,rPns); ∂ delete global rPns array record pointer;


    if typOut land debug_instruments
    then
        PRINT("GENFM1: ", CallNo," ", myproc, locPns[1]/srate, locPns[2]/srate,↓,
            tab, locPns[1], locPns[2], locPns[3], seg:name[locRec], locPns[5], ↓,
            tab, locPns[6], locPns[7], locPns[8], locPns[9], locPns[10],↓,
            tab, locPns[11], locPns[12], locPns[13],↓)
    ;

∂   Internal integer procedure GOSC(integer stsamp,nsamps| real freq|
            record_pointer(seg) ampRec|          ! This is the amplitude function|
            real ampscl,ampoff|                  ! Scale factor and offset for amplitude|
            reference itemvar fitem|             ! Returns amp rPns process item here|
            integer mod,ncs,                     ! Generator mode, number of cosines|
                fmsum,outsum                     ! FM input, output location|);

    cFq←locPns[3];
    mFq←locPns[3]*locPns[9];
    indx←locPns[11];
    peakDev←indx*mFq/2;
    maxDev←srate/4;
    normDev←peakDev/maxDev; ∂ This is given to the AmpScl term for the Mod Frq gen;
    normIndOff←(locpns[12]*mfq/2)/maxdev;

    sumMf←getsm(true); ∂ where genMf will deposit;

    genMf←GOSC(locPns[1],locPns[2],mFq,
        locFm,
        normDev,normIndOff,
        FlsegItem,
        locPns[7],locPns[8],
        zero,sumMf)
    ;

    genCf←GOSC(locPns[1],locPns[2],locPns[3],
        locRec, ∂ locPns[4] will be zero, since that's where locRec goes;
        locPns[5],locPns[6],
```

```
        ClsegItem,
        locPns[7],locPns[8],
        sumMf,locPns[13])
    ;

    Join({ClsegItem,FlsegItem});
    give(genCf);
    give(genMf);
    relsm(sumMf);
    if typOut land debug_instruments
    then
        print("    GENFM1: ",myproc, locPns[1]/srate, locPns[2]/srate,
         " Dropping off end",↓);
    rtnitm(myproc);
    end;
```

```
∂ GENFM3;

∂ sample call:
    <instrument> Beg, Dur, Freq, ampRec, Ampscl, Ampoff, Mode, Ncs,
    cfmfRatio1, indexRec1, index1, indexOff1,
    cfmfRatio2, indexRec2, index2, indexOff2,
    cfmfRatio3, indexRec3, index3, indexOff3,
    outsum;

Internal recursive procedure ins11(Record_Pointer(rpRec) rPns;
        Record_Pointer(ArrRec) Pns; integer typOut);
    begin
    itemvar F11segItem, F21segItem, F31segItem, C1segItem;
    integer genCf, genMf1, genMf2, genMf3, sumMf;
    real normDev, maxDev, mFq, cFq, normIndOff;
    real array locPns[0:21]; ∂ local array to have parameters transferred into;
    record_pointer(seg) ampRec,
        fmRec1, fmRec2, fmRec3; ∂ local pointer to the function
                                record for this inst;

    arrtran(locPns,ArrRec:X[Pns]); ∂ transfer from global Pns to local Pns;
    $RECFN(5,Pns); ∂ delete global Pns array record pointer;

    ampRec←rpRec:rpArr[rPns][4]; ∂ ampRec now points to amplitude function record;
    fmRec1←rpRec:rpArr[rPns][10]; ∂ fmRec1 is for the fm index function 1;
    fmRec2←rpRec:rpArr[rPns][14]; ∂ likewise for index function 2;
    fmRec3←rpRec:rpArr[rPns][18]; ∂ likewise for index function 3;
    $RECFN(5,rPns); ∂ delete global rPns array record pointer;


    if typOut land debug_instruments
    then
        PRINT("GENFM3: ", CallNo," ", myproc, locPns[1]/srate, locPns[2]/srate,↓,
            tab, locPns[1], locPns[2], locPns[3],↓,
            tab, seg:name[ampRec], locPns[5], locPns[6], locPns[7], locPns[8],↓,
            tab, locPns[9],  seg:name[fmRec1], locPns[11], locPns[12],↓,
            tab, locPns[13], seg:name[fmRec2], locPns[15], locPns[16],↓,
            tab, locPns[17], seg:name[fmRec3], locPns[19], locPns[20],↓,
            tab, locPns[21],↓)
    ;

∂   Internal integer procedure GOSC(integer stsamp,nsamps| real freq|
            record_pointer(seg) ampRec|          ! This is the amplitude function|
            real ampscl,ampoff|                  ! Scale factor and offset for amplitude|
            reference itemvar fitem|             ! Returns amp rPns process item here|
            integer mod,ncs,                     ! Generator mode, number of cosines|
                fmsum,outsum                     ! FM input, output location|);

    maxDev←srate/4;
    sumMf←getsm(true); ∂ where genMf will deposit;

    mFq←locPns[3]*locPns[9];
    normDev←(locPns[11]*mFq/2)/maxdev;
    normIndOff←(locpns[12]*mfq/2)/maxdev;
    genMf1←GOSC(locPns[1],locPns[2],mFq,
        fmRec1,
        normDev,normIndOff,
        F11segItem,
        locPns[7],locPns[8],
        zero,sumMf)
    ;
```

```
    mFq←locPns[3]*locPns[21];
    normDev←(locPns[15]*mFq/2)/maxdev;
    normIndOff←(locpns[16]*mfq/2)/maxdev;
    genMf1←GOSC(locPns[1],locPns[2],mFq,
        fmRec2,
        normDev,normIndOff,
        F2segItem,
        locPns[7],locPns[8],
        zero,sumMf)
    ;

    mFq←locPns[3]*locPns[17];
    normDev←(locPns[19]*mFq/2)/maxdev;
    normIndOff←(locpns[20]*mfq/2)/maxdev;
    genMf3←GOSC(locPns[1],locPns[2],mFq,
        fmRec3,
        normDev,normIndOff,
        F3segItem,
        locPns[7],locPns[8],
        zero,sumMf)
    ;

    genCf←GOSC(locPns[1],locPns[2],locPns[3],
        ampRec,
        locPns[5],locPns[6],
        C1segItem,
        locPns[7],locPns[8],
        sumMf,locPns[13])
    ;

    Join({C1segItem,F1segItem,F2segItem,F3segItem});
    give(genCf);
    give(genMf1);
    give(genMf2);
    give(genMf3);
    relsm(sumMf);
    if typOut land debug_instruments
    then
        print("    GENFM3: ",myproc, locPns[1]/srate, locPns[2]/srate,
         " Dropping off end",↓);
    rtnitm(myproc);
    end;
```

```
∂ NGEN;
∂ sample call:
    <instrument> Beg, Dur, Freq, ampRec, Ampscl, Ampoff, Mod, Ncs, Fmsum, outsum,
    #gens;

Internal recursive procedure insl2(Record_Pointer(rpRec) rPns;
        Record_Pointer(ArrRec) Pns; integer typOut);

    begin
    itemvar lsegItem;
    integer gen;
    real array locPns[0:11]; ∂ local array to have parameters transferred into;
    record_pointer(seg) locRec; ∂ local pointer to the function
                                  record for this inst;

    arrtran(locPns,ArrRec:X[Pns]); ∂ transfer from global Pns to local Pns;
    $RECFN(5,Pns); ∂ delete global Pns array record pointer;

    locRec←rpRec:rpArr[rPns][4]; ∂ locRec now points to function record;
    $RECFN(5,rPns); ∂ delete global rPns array record pointer;

∂   Internal integer procedure GOSC(integer stsamp,nsamps| real freq|
            record_pointer(seg) ampRec|          ! This is the amplitude function|
            real ampscl,ampoff|                  ! Scale factor and offset for amplitude|
            reference itemvar fitem|             ! Returns amp rPns process item here|
            integer mod,ncs,                     ! Generator mode, number of cosines|
                fmsum,outsum                     ! FM input, output location|);

    if typOut land debug_instruments
    then
        PRINT("GOSC: ", CallNo, myproc, locPns[1]/srate, locPns[2]/srate,↓,
            tab, locPns[1], locPns[2], locPns[3], seg:name[locRec], locPns[5], ↓,
            tab, locPns[6], locPns[7],locPns[8],locPns[9],locPns[10],locPns[11],↓)
    ;

    GEN←GOSC(locPns[1],locPns[2],locPns[3],
        locRec, ∂ locPns[4] will be zero, since that's where locRec goes;
        locPns[5],locPns[6],
        lsegItem,
        locPns[7],locPns[8],
        locPns[9],locPns[10])
    ;
    Join({lsegItem});
    give(gen);
    if typOut land debug_instruments
    then
        print("    GOSC: ",myproc, locPns[1]/srate, locPns[2]/srate,
         " Dropping off end",↓);
    rtnitm(myproc);
    end;
```

```
Internal recursive procedure insl3(Record_Pointer(rpRec) rPns;
        Record_Pointer(ArrRec) Pns; integer typOut);
        return;
Internal recursive procedure insl4(Record_Pointer(rpRec) rPns;
        Record_Pointer(ArrRec) Pns; integer typOut);
        return;
Internal recursive procedure insl5(Record_Pointer(rpRec) rPns;
        Record_Pointer(ArrRec) Pns; integer typOut);
        return;
```

```
END "INSTS";
```

Name: Gareth Loy

Project: 1 Programmer: DGL

File Name: SAMTBL.SAI[SAM,DGL]

File Last Written: 13:43 15 Nov 1977

Time: 22:07 Date: 16 Nov 1977

Stanford University
Artificial Intelligence Laboratory
Computer Science Department
Stanford, California

```
COMMENT ⊗   VALID 00002 PAGES
C REC  PAGE   DESCRIPTION
C00001 00001
C00002 00002    ∂ Initialization of tables
C00007 ENDMK
C⊗;
```

```
∂ Initialization of tables;

define templatelen="16";
preload_with "GOSC","GOSCVF","SHAPE","DUMMY","COSC";
string array tplids[0:templatelen];

define inslen="127";
string array insids[0:inslen]; ∂ String names of instruments;
integer array inslst[0:inslen]; ∂ Address of template for that instrument;

∂ First 12 are A,AS,B,...GS.  then SRATE,MAG;
define varlen="128";
preload_with
    440,466.16,493.89,261.62,277.18,293.66,311.13,329.63,349.23,369.99,391.99,415.31,
    25600,1.984496↑-2,0,0,
    0,0,
    4,-1,96,32,3;
real array varlst[0:varlen];
preload_with
    "A","AS","B","C","CS","D","DS","E","F","FS","G","GS",
    "SRATE","MAG","BOXTYP","RCDFLG",
    "DEBUG","NOPRINT",
    "NOUTCHANS","WHICHSIDE","NPTIX","NUTIX","FILTERS";
string array varids[0:varlen];

define sclen="49";
preload_with
    "INACTIVE","PAUSE","A_RUNNING","B_RUNNING","WAIT","C_RUNNING",
    "DATA_READ","DATA_WRITE","DAC_WRITE",
    "LPLUSQ","LMINUSQ","LEXPPLUS","LEXPMINUS",
    "SUM_OF_COSINES","SAWTOOTH","SQUARE","PULSE_TRAIN","SINE","SIN_FM",
    "SAMDEV","FRMDEV",
    "TRUE","FALSE",
    "M_INACTIVE", "U_NOISE", "TR_U_NOISE", "LATCH", "DELAY", "NOTWOPOLES",
    "TWO_0POLES", "TWO_1POLES", "NOTWOZEROES", "TWO_0ZEROES", "TWO_1ZEROES",
    "INT_MIXING", "ONE_POLE", "MIXING", "ONE_ZERO", "FOUR_QUAD_MULTIPLY",
    "MINIMUM", "MAXIMUM", "SIGNUM", "ZERO_CROSSING_PULSER",
    "ADD_SUM_MEMORY", "REPLACE_SUM_MEMORY",
    "INVOKE_DELAY_UNIT", "D_INACTIVE",
    "DELAYLINE", "TABLE_LOOKUP", "ROUND_TABLE_LOOKUP";
string array scids[0:sclen];
preload_with
    g_inactive, g_pause, a_running, b_running, g_wait, c_running, data_read,
    data_write, dac_write, lplusq, lminusq, lexpplus, lexpminus,
    sum_of_cosines, sawtooth, square, pulse_train, sine, sin_fm,
    samdev,frmdev,
    true,false,
    m_inactive, u_noise, tr_u_noise, latch, delay, notwopoles, two_0poles,
    two_1poles, notwozeroes, two_0zeroes, two_1zeroes, int_mixing, one_pole,
    mixing, one_zero, four_quad_multiply, minimum, maximum, signum,
    zero_crossing_pulser, add_sum_memory, replace_sum_memory,
    invoke_delay_unit, d_inactive,
    delayline, table_lookup, round_table_lookup;
real array sclst[0:sclen];

define smlen="1";
preload_with
    "GEN_SUM",
    "MOD_SUM";
string array smids[0:smlen];
preload_with
```

```
    True, ∂ this is what interm.sai:getsm expects if you want to alloc. gen_sum_mem;
    False;
real array smlst[0:smlen];

preload_with
    "OUTA","OUTB","OUTC","OUTD","OUTE","OUTF","OUTG","OUTH";
string array outNids[0:maxchns-1];

preload_with
    "OUTMA","OUTMB","OUTMC","OUTMD","OUTME","OUTMF","OUTMG","OUTMH";
string array outMids[0:maxchns-1];

define paklen = "2";
preload_with
    "RIGHT_JUSTIFIED","LEFT_JUSTIFIED","FULL_WORD";
string array pakIds[0:paklen];
preload_with
    right_justified,left_justified,full_word;
integer array paklst[0:paklen];
```