```
    typedef double XFLOAT;
    typedef double OTA_FLOAT;

using namespace std;

PreAlignment::PreAlignment(XFLOAT const* refBuff, long numRefSamples, XFLOAT const*
degBuff, long numDegSamples,
                           int appType, int samplingFreq, char const* outputFile, FILE*
pLogFile, void* mh)
: mFNameRef             (NULL),
  mFNameDeg             (NULL),
  mFNameOutput          (outputFile),
  mAppType              ((appType%100) % 10),
  mRef                  (NULL),
  mDeg                  (NULL),
  mAligner              (NULL),
  mResamplingResult     (NULL),
  mSegListTraversal     (NULL),
  mSamplingFreq         (samplingFreq),
  mMaxSigLenBuff        (NULL),
  matHandle             (NULL),
  mpLogFile             (NULL)
{
    OPTTRY
    {
        if (mh == NULL)
            OPTTHROW(( string("MathLib-Handle == NULL.")));

        matHandle = (MAT_HANDLE)mh;
        mpLogFile = pLogFile;

        if (mAppType < kNarrowBand || mAppType > kSuperWideBand)
            OPTTHROW(( SQError(SQERR_UNSUPPORTED_APPTYPE,"Invalid application type in
input parameters.")));
        if (samplingFreq < MIN_SAMPLING_RATE || samplingFreq > MAX_SAMPLING_RATE)
            OPTTHROW(( SQError(SQERR_SAMPLING_FREQ,"Unsupported sampling frequency for
reference and/or degraded file.")));

        int iRetVal = ReadData(refBuff, numRefSamples, degBuff, numDegSamples);
        if (iRetVal < 0)
            OPTTHROW((  SQError(iRetVal, "Error while reading data.")));

        iRetVal = Run();
        if (iRetVal != SQ_NO_ERRORS)
            OPTTHROW((  SQError(iRetVal, "Error PreAlignment time-domain
calculations.")));
    }

    OPTCATCH ((SQError err))
    {
        delete mRef;
        delete mDeg;
        delete mAligner;
        delete mResamplingResult;
        delete mSegListTraversal;
        matFree(mMaxSigLenBuff);
        OPTTHROW ((SQError(err.ErrCode(),
            "Could not start PreAlignment algorithm.\nThe following error message was
returned: "
            + err.ErrMsg() + "\n")));

    }
    OPTCATCH( (...) )
    {
        delete mRef;
        delete mDeg;
        delete mAligner;
        delete mResamplingResult;
        delete mSegListTraversal;
        matFree(mMaxSigLenBuff);
        OPTTHROW (( SQError(SQERR_OTHER, "Could not start PreAlignment algorithm because
of an unknown error.\n") ));
    }
}
```

```
PreAlignment::~PreAlignment()
{
    delete   mRef;
    delete   mDeg;
    delete   mAligner;
    delete   mResamplingResult;
    delete   mSegListTraversal;
    matFree(mMaxSigLenBuff);

    mRef               = mDeg      = NULL;
    mAligner           = NULL;
    mResamplingResult  = NULL;
    mSegListTraversal  = NULL;
    mMaxSigLenBuff     = NULL;
}

TA_SegList const* PreAlignment::GetMergedSegList() const
{
    if (mAligner == NULL)
        return NULL;

    return mAligner->MergedSegments();
}

XFLOAT PreAlignment::GetResamplingFac() const
{
    if (mAligner == NULL || mResamplingResult == NULL)
        return (XFLOAT)(-1.0);

    return mResamplingResult->fMeanResamplingFac;
}

XFLOAT PreAlignment::GetDegSNR() const
{
    if (mAligner == NULL)
        return (XFLOAT)(-1.0);

    return mAligner->SNRDeg();
}

XFLOAT PreAlignment::GetMatchQuality() const
{
    if (mAligner == NULL)
        return (XFLOAT)(-1.0);

    return mAligner->MatchQuality();
}

bool PreAlignment::ExtremeMatchFound() const
{
    if (mAligner == NULL)
        return false;

    return mAligner->ExtremeMatchFound();
}

int PreAlignment::Run()
{
    stringstream errorStream;
    clock_t startTime = clock();
    int iRetVal = SQ_NO_ERRORS;

    mTraversalVec.clear();

    iRetVal = SanityCheck();
    if (iRetVal < 0)
    {
        switch(iRetVal)
        {
        case SQERR_CORRUPTED_FILE:    errorStream << "Reference and/or degraded file(s)
too short. They may have not been properly recorded.\n"; break;
        case SQERR_REF_FILE_TOO_LONG: errorStream << "Reference file is too long. It
should be shorter than " << MAX_SPEECH_DURATION << " seconds.\n"; break;
        case SQERR_DEG_FILE_TOO_LONG: errorStream << "Degraded file is too long. It
should be shorter than " << MAX_SPEECH_DURATION << " seconds.\n"; break;
```

```
        default:                        errorStream << "Unknown error.\n"; break;
        }
        OPTTHROW ((SQError(iRetVal, errorStream.str())));

    }

    if (iRetVal == SQ_NO_ERRORS)
        iRetVal = Preprocess();

    if (iRetVal == SQ_NO_ERRORS)
        iRetVal = AlignSignals();

    return iRetVal;
}

TraversalVecType const& PreAlignment::TraversalVec() const
{
    return mTraversalVec;
}

int PreAlignment::ReadData (XFLOAT const *refBuff, long numRefSamples,
                            XFLOAT const *degBuff, long numDegSamples)
{
    OPTTRY
    {
        if(numRefSamples > 0 && numDegSamples > 0)
        {
            mRef = new SQSignal(refBuff, numRefSamples, mSamplingFreq,
STD_BIT_RESOLUTION);
            mDeg = new SQSignal(degBuff, numDegSamples, mSamplingFreq,
STD_BIT_RESOLUTION);
        }
        else
            OPTTHROW( SQError(SQERR_CORRUPTED_FILE));
    }
    OPTCATCH ((SQError err))
    {
        OPTTHROW( SQError(err.ErrCode(), "ERROR in ReadData: " + err.ErrMsg()));
    }
    OPTCATCH((string errorMsg))
    {
        OPTTHROW(SQError(SQERR_READDATA, "ERROR in ReadData: " + errorMsg));
    }
    OPTCATCH((...))
    {
        OPTTHROW(SQError(SQERR_READDATA, "ERROR in ReadData: Unknown error.\n"));
    }

    return SQ_NO_ERRORS;
}

int PreAlignment::SanityCheck()
{
    int retVal = SQ_NO_ERRORS;

    if(mRef->NrOfSamples() / (XFLOAT)mRef->SamplingFreq() < (XFLOAT)2.2)
        retVal = SQERR_CORRUPTED_FILE;

    if(mRef->NrOfSamples() / (XFLOAT)mRef->SamplingFreq() > MAX_SPEECH_DURATION)
        retVal = SQERR_REF_FILE_TOO_LONG;

    if(mDeg->NrOfSamples() / (XFLOAT)mDeg->SamplingFreq() < MIN_SPEECH_DURATION)
        retVal = SQERR_CORRUPTED_FILE;

    if(mDeg->NrOfSamples() / (XFLOAT)mDeg->SamplingFreq() > MAX_SPEECH_DURATION)
        retVal = SQERR_DEG_FILE_TOO_LONG;

    return retVal;
}

int PreAlignment::Preprocess()
{
    int iRetVal = SQ_NO_ERRORS;

    if (mRef == NULL || mDeg == NULL)
        OPTTHROW ((SQError(SQERR_PREPROC, "Cannot perform preprocessing because
```

```
reference and/or degraded signals were not created.")));
    if (mRef->IsInvalidSignal() || mDeg->IsInvalidSignal())
        OPTTHROW ((SQError(SQERR_PREPROC, "Cannot perform preprocessing because
reference and/or degraded signal contain invalid data.")));
    OPTTRY
    {
        int maxSamplingFreq = max(max(mRef->SamplingFreq(), TA_SAMPLING_RATE),
mDeg->SamplingFreq());

        int maxSigLen = max(1024, (int)ceil(
            max(mRef->NrOfSamples()/(XFLOAT)mRef->SamplingFreq()*maxSamplingFreq + 1,
                  mDeg->NrOfSamples()/(XFLOAT)mDeg->SamplingFreq()*maxSamplingFreq +
1)));
        if(mMaxSigLenBuff)
            matFree(mMaxSigLenBuff);
        mMaxSigLenBuff = (XFLOAT*)matMalloc(maxSigLen * sizeof(XFLOAT));

        if (mAppType == kSuperWideBand)
            mDeg->Preprocess(maxSamplingFreq, SQ_SIGNAL_NO_LEVEL_ALIGN,
                FRAME_LEN, FRAME_OVERLAP_RATIO, MIN_LEVEL_DB, mAppType, mMaxSigLenBuff,
0, mpLogFile);
        else
            mDeg->Preprocess(maxSamplingFreq, REF_ASL_LEVEL,
                FRAME_LEN, FRAME_OVERLAP_RATIO, MIN_LEVEL_DB, mAppType, mMaxSigLenBuff,
0, mpLogFile);

        if (mDeg->IsInvalidSignal())
            return SQERR_PREPROC;

        if (mDeg->UnalignedASL() < MIN_ASL_DEG)
            return SQERR_SPEECH_ACTIVITY;

        if ((mDeg->End() - mDeg->Start()) / (XFLOAT)mDeg->SamplingFreq() <
MIN_SPEECH_DURATION)
            return SQERR_DEG_FILE_TOO_SHORT;

        mRef->Preprocess(maxSamplingFreq,
            mDeg->CurrentASL(),
            FRAME_LEN,
            FRAME_OVERLAP_RATIO,
            MIN_LEVEL_DB, mAppType, mMaxSigLenBuff, 0, mpLogFile);

        if (mRef->IsInvalidSignal())
            return SQERR_PREPROC;

        if ((mRef->End() - mRef->Start()) / (XFLOAT)mRef->SamplingFreq() <
MIN_SPEECH_DURATION)
            return SQERR_REF_FILE_TOO_SHORT;

    }
    OPTCATCH ((string errorMsg))
    {
        OPTTHROW ((SQError(SQERR_PREPROC, "Preprocessing failed: " + errorMsg)));
    }
    OPTCATCH ((SQError err))
    {
        OPTTHROW ((SQError(err.ErrCode(), "Preprocessing failed: " + err.ErrMsg())));
    }
    OPTCATCH ((...))
    {
        OPTTHROW ((SQError(SQERR_PREPROC, "Preprocessing failed: Unknown error.\n")));
    }

    return iRetVal;
}

int PreAlignment::AlignSignals()
{
    OPTTRY
    {
        mAligner          = new SQTimeAlignment(*mRef, *mDeg, mMaxSigLenBuff, 1.0,
matHandle, mpLogFile);
        mResamplingResult = new SQTA_ResampResult();

    }
    OPTCATCH ((string errorMsg))
```

```
    {
        OPTTHROW ((SQError(SQERR_TIMEALIGNMENT, "Time alignment failed: " + errorMsg)));
    }
    OPTCATCH ((SQError err))
    {
        OPTTHROW ((SQError(err.ErrCode(), "Time alignment failed: " + err.ErrMsg())));
    }
    OPTCATCH ((...))
    {
        OPTTHROW ((SQError(SQERR_TIMEALIGNMENT, "Time alignment failed: unknown
error.")));
    }

    return SQ_NO_ERRORS;
}

int PreAlignment::TraverseSegList(XFLOAT frameLengthInSec, XFLOAT frameStepInSec)
{
    if (mRef == NULL || mDeg == NULL || mAligner == NULL ||
        mAligner->MergedSegments() == NULL || mAligner->UnusedDegSegments() == NULL)
        OPTTHROW ((string("Cannot call TraverseSegList because previous modules did not
execute successfully.")));

    OPTTRY
    {
        if (frameLengthInSec <= 0.0f || frameStepInSec <= 0.0f || frameLengthInSec <
frameStepInSec)
            OPTTHROW ((SQError(SQERR_OTHER, "Invalid frame length/step value(s).")));

        int frameLengthInSamples = round(mRef->SamplingFreq() * frameLengthInSec);
        int frameStepInSamples   = round(mRef->SamplingFreq() * frameStepInSec);
        mTraversalVec.clear();
        mSegListTraversal = new SegListTraversal(mAligner, mResamplingResult, mRef,
mDeg,
                                                 frameLengthInSamples,
frameStepInSamples);

        int curRefPos, curDegPos, minSearchPos, maxSearchPos;
        int expectedDegPos = -1;
        FRAMETYPE type;
        XFLOAT     reliability;
        bool      degActivity;
        while (mSegListTraversal->FullTraversal(curRefPos, curDegPos, minSearchPos,
maxSearchPos)
               != LEAVE_LOOP)
        {
            if (expectedDegPos < 0 && !mSegListTraversal->RefMissingInDeg())
                expectedDegPos = frameStepInSamples * (int)ceil(curDegPos /
(XFLOAT)frameStepInSamples);

            if (mSegListTraversal->SegType() == TA_SEG_MATCHED)
                degActivity = true;
            else
                degActivity =
mDeg->VADprofile()[mDeg->SamplePosToFrameNum(curDegPos+frameLengthInSamp
les/2)] == SQ_VAD_ACT_SPEECH;

            if (mSegListTraversal->RefMissingInDeg())
                type = MISSING_SPEECH;
            else if (mSegListTraversal->DegMissingInRef())
                type = INSERTED_SIG;
            else switch (mSegListTraversal->SegType())
            {
                case TA_SEG_MATCHED: type = FIXED_SPEECH;      break;
                case TA_SEG_GUESSED: type = SEARCHABLE_SPEECH; break;
                case TA_SEG_PAUSE:   type = PAUSE;             break;
                default: OPTTHROW ((string("Unexpected frame type.\n")));
            }

            reliability = mSegListTraversal->SegReliability();

            if (expectedDegPos >= 0 && mTraversalVec.size() > 0 &&
                curDegPos >= expectedDegPos + frameStepInSamples/2)
            {
                FRAMETYPE prevType = mTraversalVec.back().type;
                if (prevType != MISSING_SPEECH)
```

```
                {
                    if (prevType == PAUSE && (type == SEARCHABLE_SPEECH || type ==
FIXED_SPEECH))
                    {
                        int posDec = expectedDegPos - curDegPos;
                        mTraversalVec.push_back(TraversalStruct(curRefPos+posDec,
curDegPos+posDec,
                                                                minSearchPos+posDec,
maxSearchPos+posDec,
                                                                0.0f, degActivity,
SEARCHABLE_SPEECH));
                    }
                    else if (prevType == INSERTED_SIG)
                    {
                        int posInc = expectedDegPos - mTraversalVec.back().degPos;
                        mTraversalVec.push_back(TraversalStruct(mTraversalVec.back().ref
Pos, mTraversalVec.back().degPos+posInc,
                                                                mTraversalVec.back().min
Pos,
mTraversalVec.back().max
Pos,
                                                                0.0f,
mTraversalVec.back().deg
Activity,
mTraversalVec.back().typ
e));
                    }
                    else
                    {
                        int posInc = expectedDegPos - mTraversalVec.back().degPos;
                        int newDelay = mTraversalVec.back().refPos -
mTraversalVec.back().degPos;
                        if ((prevType == SEARCHABLE_SPEECH || prevType == FIXED_SPEECH)
&&
                            (type     == SEARCHABLE_SPEECH || type     == FIXED_SPEECH))
                            newDelay = (newDelay + (curRefPos-curDegPos)) / 2;
                        int newRefPos = mTraversalVec.back().degPos + posInc + newDelay;
                        mTraversalVec.push_back(TraversalStruct(newRefPos,
mTraversalVec.back().degPos+posInc,
                                                                min(mTraversalVec.back()
.minPos+posInc,
newRefPos),
                                                                max(mTraversalVec.back()
.maxPos+posInc,
newRefPos),
                                                                0.0f,
mTraversalVec.back().deg
Activity,
mTraversalVec.back().typ
e));
                    }
                }
                else
                {
                    int posDec = expectedDegPos - curDegPos;
                    int firstMissingFrameIdx = (int)mTraversalVec.size()-1;
                    for (; firstMissingFrameIdx > 0 &&
mTraversalVec.at(firstMissingFrameIdx).type == MISSING_SPEECH;
firstMissingFrameIdx--);
                    firstMissingFrameIdx++;
                    mTraversalVec.push_back(TraversalStruct(curRefPos+posDec,
curDegPos+posDec,
                                            min(min(minSearchPos, curRefPos+posDec),
mTraversalVec.at(firstMissingFrameIdx).refPo
s), maxSearchPos,
                                            0.0f, degActivity, type == FIXED_SPEECH ?
SEARCHABLE_SPEECH : type));
                }

                if (mTraversalVec.back().type != MISSING_SPEECH)
                    expectedDegPos += frameStepInSamples;

                mTraversalVec.push_back(TraversalStruct(curRefPos, curDegPos,
                    minSearchPos, maxSearchPos,
                    reliability, degActivity, type));
```

```
                if (!mSegListTraversal->RefMissingInDeg())
                    expectedDegPos += frameStepInSamples;
            }
            else if (expectedDegPos >= 0 && mTraversalVec.size() > 0 && type !=
MISSING_SPEECH &&
                     curDegPos < expectedDegPos - frameStepInSamples/2)
            {
                FRAMETYPE prevType = mTraversalVec.back().type;
                if (prevType != MISSING_SPEECH)
                {
                    if (prevType == PAUSE && (type == SEARCHABLE_SPEECH || type ==
FIXED_SPEECH))
                    {
                        mTraversalVec.back().minPos = min(mTraversalVec.back().minPos,
minSearchPos);
                        mTraversalVec.back().maxPos = max(mTraversalVec.back().maxPos,
maxSearchPos);
                        mTraversalVec.back().type = SEARCHABLE_SPEECH;
                        mTraversalVec.back().degActivity =
mTraversalVec.back().degActivity || degActivity;
                        mTraversalVec.back().reliability = 0.0f;
                    }
                    else if (prevType == INSERTED_SIG)
                    {
                        mTraversalVec.back().minPos = min(mTraversalVec.back().minPos,
minSearchPos);
                        mTraversalVec.back().maxPos = max(mTraversalVec.back().maxPos,
maxSearchPos);
                        mTraversalVec.back().degActivity =
mTraversalVec.back().degActivity || degActivity;
                        mTraversalVec.back().reliability = 0.0f;
                    }
                    else
                    {
                        if (type != MISSING_SPEECH)
                        {
                            mTraversalVec.back().minPos =
min(mTraversalVec.back().minPos, minSearchPos);
                            mTraversalVec.back().maxPos =
max(mTraversalVec.back().maxPos, maxSearchPos);
                        }
                        mTraversalVec.back().degActivity =
mTraversalVec.back().degActivity || degActivity;
                        mTraversalVec.back().reliability = 0.0f;
                        if (prevType == FIXED_SPEECH)
                            mTraversalVec.back().type = SEARCHABLE_SPEECH;
                    }
                }
                else
                {
                    mTraversalVec.pop_back();
                    mTraversalVec.push_back(TraversalStruct(curRefPos, curDegPos,
                                                            minSearchPos, maxSearchPos,
                                                            0.0f, degActivity, type));
                }

            }
            else
            {
                mTraversalVec.push_back(TraversalStruct(curRefPos, curDegPos,
                                                        minSearchPos, maxSearchPos,
                                                        reliability, degActivity,
type));

                if (!mSegListTraversal->RefMissingInDeg())
                    expectedDegPos += frameStepInSamples;
            }

            mSegListTraversal->MoveToNextFramePair();
        }

        delete mSegListTraversal;
        mSegListTraversal = NULL;

        XFLOAT SNR = mDeg->CurrentASL() - mDeg->CurrentNoiseLevel();
        int maxIntervalLen = limit(round((0.12*SNR - 1.4)*0.021333/frameStepInSec),
```

```
                                    1, round(4 * 0.021333/frameStepInSec));
        int silIntervStart = -1;
        for (int i = 1; i < (int)mTraversalVec.size()-1; i++)
        {
            if (mTraversalVec[i].type == PAUSE)
            {
                silIntervStart = -1;
                continue;
            }

            if (silIntervStart < 0 && mTraversalVec[i].degActivity == false)
                silIntervStart = i;
            else if (silIntervStart > 0 && mTraversalVec[i].degActivity == true)
            {
                if (i-silIntervStart <= maxIntervalLen)
                    for (int j = silIntervStart; j < i; j++)
                        mTraversalVec[j].degActivity = true;
                silIntervStart = -1;
            }
        }

    }
    OPTCATCH ((string errorMsg))
    {
        delete mSegListTraversal;
        mSegListTraversal = NULL;
        OPTTHROW ((SQError(SQERR_CORE, "Segment list traversal module failed: " +
errorMsg)));
    }
    OPTCATCH ((SQError err))
    {
        delete mSegListTraversal;
        mSegListTraversal = NULL;
        OPTTHROW ((SQError(err.ErrCode(), "Segment list traversal module failed: " +
err.ErrMsg())));
    }
    OPTCATCH ((...))
    {
        delete mSegListTraversal;
        mSegListTraversal = NULL;
        OPTTHROW ((SQError(SQERR_CORE, "Segment list traversal module failed: Unknown
error.\n")));
    }

    return SQ_NO_ERRORS;
}
```