```
    typedef double XFLOAT;
    typedef double OTA_FLOAT;

using namespace std;

void* PreAlignment_Init(XFLOAT const* refBuff, long numRefSamples,
                                XFLOAT const* degBuff, long numDegSamples,
                                int samplingFreq, int appType, FILE* pLogFile, void* mh)
{

    //Find memory leaks
    _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);
    _CrtSetReportMode(_CRT_ERROR, _CRTDBG_MODE_DEBUG);

    //Initialize WIN32 exception handler
    win32_exception::install_handler();

    int iRetVal        = SQERR_UNKNOWN_ERROR;
    PreAlignment *Algo = NULL;
    OPTTRY
    {
        OPTTRY
        {

            //Load input files and desired frame configuration
            XFLOAT frameLengthInSec = (XFLOAT)0.032;
            XFLOAT frameStepInSec   = (XFLOAT)0.016;
            Algo = new PreAlignment(refBuff, numRefSamples, degBuff, numDegSamples,
                appType, samplingFreq, 0, pLogFile, mh);

            return (void*)Algo;
        }
        catch (access_violation const &e)
        {
            stringstream errStream;
            errStream << e.what() << " at " << std::hex << e.where() << ": Bad " <<
(e.isWrite() ? "write" : "read")
                << " on " << e.badAddress() << std::endl;
            throw SQError(SQERR_OTHER, errStream.str());
        }
        catch (win32_exception  const &e)
        {
            stringstream errStream;
            errStream << e.what() << " (code " << std::hex << e.code() << ") at " <<
e.where() << std::endl;
            throw SQError(SQERR_OTHER, errStream.str());
        }
    }
    OPTCATCH ((string errorMsg))
    {

        delete Algo;
        Algo = NULL;
        return (void*)Algo;
    }
    OPTCATCH ((SQError err))
    {

        delete Algo;
        Algo = NULL;
        return (void*)Algo;
    }
    OPTCATCH ((...))
    {

        delete Algo;
        Algo = NULL;
        return (void*)Algo;
    }
}

XFLOAT PreAlignment_GetResamplingFac(void *PAHandle)
{
    if (PAHandle == NULL)
```

```
        return (XFLOAT)-1.0;

    PreAlignment *Algo = (PreAlignment*)PAHandle;
    XFLOAT fac = (XFLOAT)-1.0;

    OPTTRY
    {
        fac = Algo->GetResamplingFac();
    }
    OPTCATCH ((...))
    {
        fac = (XFLOAT)-1.0;
    }

    return fac;
}

XFLOAT PreAlignment_GetDegSNR(void *PAHandle)
{
    if (PAHandle == NULL)
        return (XFLOAT)-1.0;

    PreAlignment *Algo = (PreAlignment*)PAHandle;
    XFLOAT snr = (XFLOAT)-1.0;

    OPTTRY
    {
        snr = Algo->GetDegSNR();
    }
    OPTCATCH ((...))
    {
        snr = (XFLOAT)-1.0;
    }
    return snr;
}

XFLOAT PreAlignment_GetMatchQuality(void *PAHandle)
{
    if (PAHandle == NULL)
        return (XFLOAT)-1.0;

    PreAlignment *Algo = (PreAlignment*)PAHandle;
    XFLOAT matchQual = (XFLOAT)-1.0;

    OPTTRY
    {
        matchQual = Algo->GetMatchQuality();
    }
    OPTCATCH ((...))
    {
        matchQual = (XFLOAT)-1.0;
    }
    return matchQual;
}

bool PreAlignment_ExtremeMatchFound(void *PAHandle)
{
    if (PAHandle == NULL)
        return false;

    PreAlignment *Algo = (PreAlignment*)PAHandle;
    bool extremeMatchFound = false;

    OPTTRY
    {
        extremeMatchFound = Algo->ExtremeMatchFound();
    }
    OPTCATCH ((...))
    {
        extremeMatchFound = false;
    }
    return extremeMatchFound;
}

TA_SegList const* PreAlignment_GetSegList(void *PAHandle)
{
```

```
    if (PAHandle == NULL)
        return NULL;

    PreAlignment *Algo = (PreAlignment*)PAHandle;
    TA_SegList const *pResult = NULL;
    OPTTRY
    {
        pResult = Algo->GetMergedSegList();
    }
    OPTCATCH( (...))
    {
        pResult = NULL;
    }
    return pResult;
}

TraversalVecType const* PreAlignment_Traverse(void *PAHandle, XFLOAT frameLengthInSec,
XFLOAT frameStepInSec)
{
    if (PAHandle == NULL)
        return NULL;

    PreAlignment *Algo = (PreAlignment*)PAHandle;
    TraversalVecType const *pResult = NULL;
    OPTTRY
    {
        Algo->TraverseSegList(frameLengthInSec, frameStepInSec);
        pResult = &(Algo->TraversalVec());
    }
    OPTCATCH ((...))
    {
        pResult = NULL;
    }
    return pResult;
}

int PreAlignment_Free(void **pPAHandle)
{
    if (pPAHandle == NULL || *pPAHandle == NULL)
        return -1;

    PreAlignment *Algo = (PreAlignment*)(*pPAHandle);
    delete Algo;
    Algo = (PreAlignment*)NULL;
    *pPAHandle = NULL;

    return 0;
}
```