```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <cstdlib>
#include <vector>
#include <time.h>
#include <sstream>
#include <boost/thread.hpp>


typedef std::vector<int> SimilarityVector;
typedef std::vector<SimilarityVector> SimilarityMatrix;
typedef std::vector<std::string> SequenceVector;

timespec timespec_diff(timespec start, timespec end);
void printMatrix(const SimilarityMatrix &matrix);

struct ThreadScheduler
{
  boost::mutex mutex;
    int nextFreeThread;
    int maxThread;
};

class MultiThread
{
public:
    MultiThread(SimilarityMatrix *similarityMatrix, SequenceVector *sequences, ThreadScheduler *
threadScheduler, int row)
        :similarityMatrix(similarityMatrix),
         sequences(sequences),
         threadScheduler(threadScheduler)
    {}

  void start()
  {
    thread = boost::thread(&MultiThread::operator(), this);
  }

    void join()
    {
        thread.join();
    }

    void operator()()
    {
        int rows = similarityMatrix->size();
        int cols = (*similarityMatrix)[0].size();

        int thrnum;
    {
      boost::lock_guard<boost::mutex> lock(threadScheduler->mutex);
      thrnum = threadScheduler->nextFreeThread++;

        }

    // go until the grid is done
    while(thrnum < rows)
        {


            for (int i = 1 ; i <=cols-1; i++)
            {
                    // get cell coordinates
                int row = thrnum;
                int col = i;
```

```
 66
 67
 68                 while((*similarityMatrix)[row - 1][col + 1] < 0 && (*similarityMatrix)[row - 1][col] < 0)
 69                     ;
 70
 71                 int options[3];
 72
 73                 // Algorithm goes here
 74         options[0] = (*similarityMatrix)[row - 1][col - 1] + ((*sequences)[0][row - 1] == (*sequences)[1][
col - 1] ? 1 : -1);
 75                 options[1] = (*similarityMatrix)[row][col - 1] - 2;
 76                 options[2] = (*similarityMatrix)[row - 1][col] - 2;
 77
 78                 int value = 0;
 79                 for(int o = 0; o < 3; ++o)
 80                     if(options[o] > value)
 81                         value = options[o];
 82
 83
 84                 (*similarityMatrix)[row][col] = value;
 85
 86             }
 87
 88
 89             {
 90                 boost::lock_guard<boost::mutex> lock(threadScheduler->mutex);
 91                 thrnum = threadScheduler->nextFreeThread++;
 92             }
 93         }
 94    }
 95
 96 private:
 97   SimilarityMatrix *similarityMatrix;
 98     SequenceVector *sequences;
 99     ThreadScheduler *threadScheduler;
100     boost::thread thread;
101 };
102
103 /**
104  * Main function.
105  *
106  */
107 int main(int argc, char *argv[])
108 {
109
110     std::ifstream files[2];
111
112     files[0].open("C://Krishna/HPC-Files/Prog-Assign2/HIV-1_db.fasta");
113
114     files[1].open("C://Krishna/HPC-Files/Prog-Assign2/HIV-1_Polymerase.txt");
115
116     if(!(files[0]&&files[1]))
117     {
118         std::cerr<<"Unable To load the file";
119         exit(EXIT_FAILURE);
120     }
121
122     // read files
123     SequenceVector sequences(2);
124
125     std::string line;
126     for(int i = 0; i < 2; ++i)
127     {
128         std::ifstream &file = files[i];
129         std::string &sequence = sequences[i];
130         while(getline(file, line))
```

```
        {
            if((line.size() > 0) && (line[line.size() - 1] == '\r'))
                line.resize(line.size() - 1);
            sequence += line;
        }
    }

    // check the sequence size is >= the sample size
    int rows = sequences[0].size() + 1;
    int cols = sequences[1].size() + 1;
    if(rows < cols)
    {
        exit(EXIT_FAILURE);
    }

    // create the matrix, setting all cells to -1
    SimilarityMatrix similarityMatrix(rows, SimilarityVector(cols, -1));

    // set diagonal thread scheduling helpers
    ThreadScheduler threadScheduler;
    threadScheduler.nextFreeThread = 1;
    threadScheduler.maxThread = rows ;
    // set the first row and and first column to 0
    for(int r = 0; r < rows; ++r)
        similarityMatrix[r][0] = 0;
    for(int c = 0; c < cols; ++c)
        similarityMatrix[0][c] = 0;

    // start timing
    struct timespec start, finish;
    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &start);
    // trigger the threads
    std::vector<MultiThread*> threads;

    for(int threadIndex = 0; threadIndex < 4; ++threadIndex)
    {
        MultiThread *t = new MultiThread(&similarityMatrix, &sequences, &threadScheduler, threadIndex);
        threads.push_back(t);
        t->start();
    }

    // wait for the threads to finish
    for(std::vector<MultiThread*>::iterator it = threads.begin(); it != threads.end(); ++it)
    {
        (*it)->join();
        delete *it;
    }

    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &finish);

    printMatrix(similarityMatrix);

std::cerr << "Total time (nanoseconds): " << timespec_diff(start, finish).tv_nsec << std::endl;

  return 0;
}


void printMatrix(const SimilarityMatrix &matrix)
{
    for(SimilarityMatrix::const_iterator rowit = matrix.begin(); rowit != matrix.end(); ++rowit)
    {
        for(SimilarityVector::const_iterator colit = rowit->begin(); colit != rowit->end(); ++colit)
            std::cout << std::setw(3) << *colit << ' ';
        std::cout << '\n';
    }
```

```
197     std::cout << std::flush;
198 }
199
200
201
202
203 timespec timespec_diff(timespec start, timespec end)
204 {
205     timespec temp;
206     if ((end.tv_nsec - start.tv_nsec) < 0)
207     {
208         temp.tv_sec = end.tv_sec - start.tv_sec-1;
209     temp.tv_nsec = 1000000000 + end.tv_nsec - start.tv_nsec;
210   }
211     else
212     {
213     temp.tv_sec = end.tv_sec - start.tv_sec;
214     temp.tv_nsec = end.tv_nsec - start.tv_nsec;
215   }
216   return temp;
217 }
```