```
//File Name: tapestreamer.h
//Author: Rune Saetre
/*Description: Library functions for the tapestreamer*/
//Last Changed: February 26, 1997
//————————————————————————

#ifndef TAPESTREAMER_H
#define TAPESTREAMER_H

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <sys/mtio.h>
#include <sys/file.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <sys/scsi/impl/uscsi.h>
#include <iostream.h>
#include <sys/scsi/generic/commands.h>

#include "uscsiclass.h"

#define BLOCKSIZE 32768
/* #define BLOCKSIZE 1024 */

#define FYSISK 0x04
#define LOGISK 0x00


class TapeStreamer
{
  int valid_status;
  UserSCSI *scsi;
  Error *error;

  public:

  TapeStreamer();
  TapeStreamer(const char *fname);      /* Tek namnet p tapedev som parameter */
  ~TapeStreamer();

  int read_posi(long *pos);      /* Legg posisjon i *pos */
  int goto_posi(long *pos,      /* Gjeng til *pos */
                unsigned char postype);      /* LOGISK for logisk,
```

```
*SCSI::send_cmd ved feil*/


  int space(long count);       /* skjer count blokker framover. */

  int read_scsi_status();      /* Ikkje implementert.
                                                    Krev ein del lesing... */

  int goto_eom();
  int rewind();
  int unload();

  int inquiry(char *data);      // Det maa vere sett til sides 256 bytes til data.

  int set_no_compression();

/* Br ikkje brukast */
  int read_32k(char *data);     /* Les 32KB    */
  int write_32k(char *data);    /* Skriv 32KB */

/* Desse br brukast i staden */
  int read_block(char *data);     /* Les ei logisk blokk */
  int write_block(char *data);    /* Skriv ei logisk blokk */
                                  /* (Begge desse nyttar BLOCKSIZE) */

  /* Kopierar stasjons-ID-en til id */
  int streamer_id(char *id);
};

#endif      /* TAPESTREAMER_H */
```

```
//File Name: write_log.cc
//Author: Rune Saetre & Thomas Maukon Andersen
/*Description: Writes blocks to the tapestreamer and
  dumps the writetimes to stdout.
  Measures the total time to write a tape*/
//Last Changed: September 20, 1997
//--------------------

#include "../station/streamer/tapestreamer.h"
#include <stdio.h>
#include <time.h>

// Reads blocks from the tapestreamer and dumps them on stdout.
main()
{

  const char *tapename = getenv("TAPE");
  if(tapename == NULL)
        tapename = "/dev/rmt/0hn";

  // Create tapestreamer-object
  TapeStreamer *streamer;
  streamer = new TapeStreamer(tapename);

  // Rewind and disable compression
  streamer->set_no_compression();
  streamer->rewind();

  // Space for data
  unsigned char data[32][BLOCKSIZE];

  // The time
  time_t starttime;
  time_t stoptime;
  long msecs, oldmsecs, d_msecs = 0;
  struct timeval time_, *tp;
  tp = &time_;

  // The position
  long posi = 0, *pos = &posi;

  // Initialize the data to random values
  for(int i=0; i<32; i++)
        for(int j=0; j<BLOCKSIZE; j++)
              data[i][j] = (unsigned char) rand();

  // Get starttime for writing
  starttime = time(NULL);
```

```
// write some sort of header
cout << "Block time written position\n";

// Initialize time and write first block
gettimeofday(tp, NULL);
oldmsecs = ((tp->tv_sec%1000000)*1000) + (tp->tv_usec/1000);

int ret = streamer->write_block((char *)data[0]);

gettimeofday(tp, NULL);
msecs = ((tp->tv_sec%1000000)*1000) + (tp->tv_usec/1000);
d_msecs = msecs - oldmsecs;

streamer->read_posi(pos);

cout << "0 " << d_msecs << " " << ret << " " << posi << endl;

int i = 0;      // Variable to select block to write
while(ret > 0)
        {
                i++;

                // Get time
                gettimeofday(tp, NULL);
                oldmsecs = ((tp->tv_sec%1000000)*1000) + (tp->tv_usec/1000);

                // Write block
                ret = streamer->write_block((char *)data[i%32]);

                // Get time and calculate difference
                gettimeofday(tp, NULL);
                msecs = ((tp->tv_sec%1000000)*1000) + (tp->tv_usec/1000);
                d_msecs = msecs - oldmsecs;

                // Read position
                streamer->read_posi(pos);

                // Write everything to stdout
                cout << i << " " << d_msecs << " " << ret
                        << " " << posi << endl;

        }

// An error must have occurred
delete streamer;

// Get finish-time and calculate time spent
```

```
  stoptime = time(NULL);
  stoptime = stoptime - starttime;

  cout << "Total time to write tape (s):  "
       << stoptime << endl;
  cout << "Which is " << (stoptime/60) << " mins and "
       << stoptime%60 << "secs\n";
  cout << "Or " << (stoptime/3600.0) << " hours.\n";
}
```

```
//File Name: trekk.cc
//Author: Thomas Maukon Andersen
/*Description: Draws random accesses in a user defined
  interval. Writes the posistion and segment length
  to stdout*/
//Last Changed: October 3, 1997
//--------------------

#include <stdio.h>
#include <iostream.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>

/* ##### IMPORTANT!!! #####
   Input to the function must be written in this order:
   length : number of blocks to be read
   no_of_draws : number of samples to be drawn
   startpos : lowest number to be drawn
   stoppos : highest number to be drawn
   */

main(int argc, char *argv[])
{
  sleep(2);      // Make sure that time(NULL) is different every time
  srand48(time(NULL));      // Initialize random generator

  long no_of_draws = 128;
  long startpos = 0;
  long stoppos = 385000;
  long length = 1;
  long position, readlength;

  // Get in-parameters
  if(argc >= 2)
    length = atol(argv[1]);      // Max. size of the segment to be read
  if(argc >= 3)
    no_of_draws = atol(argv[2]);      // Number of samples to be drawn
  if(argc >= 4)
    startpos = atol(argv[3]);      // Lowest position
  if(argc >= 5)
    stoppos = atol(argv[4]);      // Highest position

  if (startpos > stoppos)
    {
      cout << "Illegal input!  Startpos > Stoppos\n";
      exit(1);
    }
```

```
    // Header
    cout << "position length\n";

    // Draw!
    for (int i=1; i<=no_of_draws; i++)
            {
                    position = lrand48()%(stoppos-startpos)+startpos;
                    readlength = lrand48()%length+1;
                    cout << position << " " << readlength << endl;
            }
}
```

```
//File Name: SORT.cc
//Author: Thomas Maukon Andersen
/*Description: Takes a file generated with trekk.cc
  and reorders the accesses in increasing order
  using the INSERTION SORT algorithm. Writes the
  reordered list to stdout*/
//Last Changed: October 8, 1997
//--------------------

#include "../station/streamer/tapestreamer.h"
#include <stdio.h>
#include <iostream.h>
#include <stdlib.h>
#include <fstream.h>

main(int argc, char *argv[])
{
  // Starting time
  long starttime, stoptime;
  struct timeval time, *tp;
  tp = &time;

  gettimeofday(tp, NULL);
  starttime = ((tp->tv_sec%1000000)*1000)+(tp->tv_usec/1000);

  // General variables
  const char *filename = "no_file_specified";
  char thrash[16];
  int sort_amount = 16;

  // Structure to hold blocknumbers and readlength
  struct request
  {
    long addr;      // Blocknumber to start reading from
    int readlen;     // Number of 32kb blocks to be read
  };

  // Array to hold the data.
  request tor[2048];      // tor = table of requests :)

  // Get in-parameters
  if(argc >= 2)
    filename = argv[1];
  if(argc >= 3)
    sort_amount = atol(argv[2])-1;      // To be used as index in array

  // Open the datafile
  ifstream innfil;
```

```
  innfil.open(filename);
  if (innfil.fail())
    {
      cout << "Kan ikke aapne innfil <" << filename << ">\n";
      exit(1);
    }

  // Read the first line (the header)
  innfil >> thrash;
  innfil >> thrash;

  // Put blocknumbers and readlength from file into array
  for (int i=0; i<=sort_amount; i++)
    {
      innfil >> tor[i].addr;
      innfil >> tor[i].readlen;
    }

  // Sort the table on blocknumbers using insertion sort
  request key;
  int k;
  for (int j=1; j<=sort_amount; j++)
    {
      key = tor[j];
      k = j-1;
      while ((k > -1) && (tor[k].addr > key.addr))
        {
          tor[k+1] = tor[k];
          k--;
        }
      tor[k+1] = key;
    }

  // Get the time and calculate the time spent
  gettimeofday(tp, NULL);
  stoptime = ((tp->tv_sec%1000000)*1000)+(tp->tv_usec/1000);
  stoptime = stoptime - starttime;
  cout << "Time_used:   " << stoptime << endl;

  // Write out the sorted list
  for (int m=0; m<=sort_amount; m++)
    cout << tor[m].addr << "  " << tor[m].readlen
         << " 1\n";      // The "1" is only to preformat for run_draw128.cc

  //cleanup
  innfil.close();
}
```

```
//File Name: NSCAN.cc
//Author: Thomas Maukon Andersen
/*Description: Takes a file generated with trekk.cc
  and reorders the accesses according to the
  N-SCAN-algorithm. Writes the reordered list to
  stdout. */
//Last Changed: October 20, 1997
//--------------------

#include "../station/streamer/tapestreamer.h"
#include <stdio.h>
#include <iostream.h>
#include <stdlib.h>
#include <fstream.h>

main(int argc, char *argv[])
{
  // Starting time
  long starttime, stoptime;
  struct timeval time, *tp;
  tp = &time;
  int N = 5537;

  gettimeofday(tp, NULL);
  starttime = ((tp->tv_sec%1000000)*1000)+(tp->tv_usec/1000);

  // General variables
  const char *filename = "no_file_specified";
  char thrash[16];
  int scan_amount = 16;

  // Structure to hold blocknumbers and readlength
  struct request
  {
    long addr;      // Blocknumber to start reading from
    int readlen;     // Number of 32kb blocks to be read
    int forward;     // 1 if forward, 0 if rewerse track
    int pos;      // calculated relative position
  };

  // Array to hold the data.
  request tor[2048];      // tor = table of requests :)

  // Get in-parameters
  if(argc >= 2)
    filename = argv[1];
  if(argc >= 3)
    N = atol(argv[2]);
```

```
if(argc ≥ 4)
  scan_amount = atol(argv[3])-1;      // To be used as array index

// Open the datafile
ifstream innfil;
innfil.open(filename);
if (innfil.fail())
  {
    cout ≪ "Kan ikke aapne innfil <" ≪ filename ≪ ">\n";
    exit(1);
  }

// Read the first line (the header)
innfil ≫ thrash;
innfil ≫ thrash;

// Put blocknumbers and readlength from file into array
// and calculate physical position
long signed_pos;
for (int i=0; i≤scan_amount; i++)
  {
    innfil ≫ tor[i].addr;
    innfil ≫ tor[i].readlen;

    // The physical position (signed)
    signed_pos = ((tor[i].addr+N)%(2*N))-N;

    if (signed_pos ≥ 0)
      tor[i].forward = 1;
    else
      tor[i].forward = 0;

    tor[i].pos = labs(signed_pos);
  }

// Sort the table according to the SCAN algorithm
// using a variant of insertion sort
request key;
int k;

for (int j=1; j≤scan_amount; j++)
  {
    key = tor[j];
    k = j-1;
    if (key.forward > 0)
      {
        while ((k > -1) &&
               ((tor[k].pos > key.pos) || (tor[k].forward == 0)))
```

```
            {
              tor[k+1] = tor[k];
              k--;
            }
          tor[k+1] = key;
        }

      else      // key.forward == 0
        {
          while ((k > -1) &&
                 (tor[k].pos < key.pos) && (tor[k].forward == 0))
            {
              tor[k+1] = tor[k];
              k--;
            }
          tor[k+1] = key;
        }
    }

  // Get the time and calculate the time spent
  gettimeofday(tp, NULL);
  stoptime = ((tp→tv_sec%1000000)*1000)+(tp→tv_usec/1000);
  stoptime = stoptime - starttime;

  cout ≪ "Time_used:   " ≪ stoptime ≪ endl;

  // Write out the sorted list
  for (int m=0; m≤scan_amount; m++)
    cout ≪ tor[m].addr ≪ "  " ≪ tor[m].readlen
         ≪ "  " ≪ tor[m].pos ≪ endl;

  // Cleanup
  innfil.close();
}
```

```
//File Name: KSCAN.cc
//Author: Thomas Maukon Andersen
/*Description: Takes a file generated with trekk.cc
  and reorders the accesses according to the
  K-SCAN-algorithm.
  Uses the 'write-turn' algorithm to characterize
  the tape.
  Writes the reordered list to stdout. */
//Last Changed: October 22, 1997
//--------------------

#include "../station/streamer/tapestreamer.h"
#include <stdio.h>
#include <iostream.h>
#include <stdlib.h>
#include <fstream.h>

main(int argc, char *argv[])
{
  // Starting time
  long starttime, stoptime;
  struct timeval time, *tp;
  tp = &time;

  gettimeofday(tp, NULL);
  starttime = ((tp->tv_sec%1000000)*1000)+(tp->tv_usec/1000);

  // General variables
  const char *requestfile = "no_file_specified";
  const char *bottomfile = "no_file_specified";
  char thrash[16];
  int scan_amount = 16;

  // Structure to hold blocknumbers and readlength
  struct request
  {
    long addr;      // Blocknumber to start reading from
    int readlen;     // Number of 32kb blocks to be read
    int forward;     // 1 if forward, 0 if rewerse track
    int pos;     // calculated relative position
  };

  // Structure to hold data about each trackset
  struct trackinfo
  {
    int N;     // # of blocks on each trackset
    long akk_N;     // Akkumulated # of blocks on this + previous blk.sets
  };
```

```
// Array to hold the request data.
request tor[2048];      // tor = table of requests :)

// Array to hold the characterising data
trackinfo tracktable[72];

// Get in-parameters
if(argc ≥ 2)
  requestfile = argv[1];
if(argc ≥ 3)
  bottomfile = argv[2];
if(argc ≥ 4)
  scan_amount = atol(argv[3])-1;

// Open the requestfile
ifstream req_file;
req_file.open(requestfile);
if (req_file.fail())
  {
    cout ≪ "Kan ikke aapne req_file <" ≪ requestfile ≪ ">\n";
    exit(1);
  }

// Open the bottomfile
ifstream end_of_tracks;
end_of_tracks.open(bottomfile);
if (end_of_tracks.fail())
  {
    cout ≪ "Kan ikke aapne end_of_tracks <" ≪ bottomfile ≪ ">\n";
    exit(1);
  }

// Putting characterising data into tracktable
end_of_tracks ≫ tracktable[0].akk_N;
end_of_tracks ≫ thrash;      // Second coloumn
tracktable[0].N = tracktable[0].akk_N;

for (int i=1; i≤71; i++)
  {
    end_of_tracks ≫ tracktable[i].akk_N;
    end_of_tracks ≫ thrash;      // Second coloumn
    tracktable[i].N = tracktable[i].akk_N - tracktable[i-1].akk_N;
  }

// Put blocknumbers and readlength from file into request array
// Read the first line (the header)
req_file ≫ thrash;
```

```
req_file ≫ thrash;

for (int i=0; i≤scan_amount; i++)
  {
    req_file ≫ tor[i].addr;
    req_file ≫ tor[i].readlen;

    if (tor[i].addr < tracktable[0].akk_N)       // Position on 1st trackset
      {
        tor[i].pos = tor[i].addr;
        tor[i].forward = 1;
      }

    else if (tor[i].addr > tracktable[71].akk_N)       // Illegal position
      {
        cout ≪ "Illegal request:   " ≪ tor[i].addr ≪ " Exiting\n";
        exit(1);
      }

    else      // Repeat until correct trackset is found
      {
        int a = 1;
        while (!((tor[i].addr > tracktable[a-1].akk_N) &&
                 (tor[i].addr ≤ tracktable[a].akk_N)))
          {
            a++;
          }
        tor[i].pos = tor[i].addr - tracktable[a-1].akk_N;

        // Decide if on forward or reverse trackset
        if ((a-1)%2 == 0)
          tor[i].forward = 0;
        else
          tor[i].forward = 1;
      }
  }

// Sort the table according to the SCAN algorithm
// using a variant of insertion sort
request key;
int k;

for (int j=1; j≤scan_amount; j++)
  {
    key = tor[j];
    k = j-1;
    if (key.forward > 0)
      {
```

```
            while ((k > -1) &&
                    ((tor[k].pos > key.pos) || (tor[k].forward == 0)))
              {
                tor[k+1] = tor[k];
                k--;
              }
            tor[k+1] = key;
          }

        else      // key.forward == 0
          {
            while ((k > -1) &&
                    (tor[k].pos > key.pos) && (tor[k].forward == 0))
              {
                tor[k+1] = tor[k];
                k--;
              }
            tor[k+1] = key;
          }
      }

    // Get the time and calculate the time spent
    gettimeofday(tp, NULL);
    stoptime = ((tp→tv_sec%1000000)*1000)+(tp→tv_usec/1000);
    stoptime = stoptime - starttime;

    cout ≪ "Time_used:   " ≪ stoptime ≪ endl;

    // Write out the sorted list
    for (int m=0; m≤scan_amount; m++)
      cout ≪ tor[m].addr ≪ "  " ≪ tor[m].readlen
           ≪ "  " ≪ tor[m].pos ≪ endl;

    // Cleanup
    req_file.close();
    end_of_tracks.close();
}
```

```
//File Name: SLTF.cc
//Author: Thomas Maukon Andersen
/*Description: Takes a file generated with trekk.cc
  and reorders the accesses according to the
  SLTF-algorithm.
  Uses the 'write-turn' algorithm to characterize
  the tape.
  Writes the reordered list to stdout. */
//Last Changed: December 16, 1997
//--------------------

#include "../station/streamer/tapestreamer.h"
#include <stdio.h>
#include <iostream.h>
#include <stdlib.h>
#include <stddef.h>
#include <fstream.h>
#include <math.h>

const int T_0 = 2000;
const long T_WIND = 120000;
const int T_REW = 2000;
const int T_SAW = 4000;
const long SLT_INIT = 240000;
const int TOOTH = 200;

// Structure to hold request-data
struct request
{
  long addr;      // Blocknumber to start reading from
  int readlen;      // Number of 32kb blocks to be read
  int pos;      // calculated relative position
  int track;      // Track corresponding to addr and bottomfile
  long loctime;      // locatetime from previous access
  request *prev;      // Pointer to the previous struct element
  request *next;      // Pointer to next struct element
};
typedef request* requestptr;      // Linked list

// Structure to hold data about each trackset
struct trackinfo
{
  int N;      // # of blocks on each trackset
  long akk_N;      // Akkumulated # of blocks on this + previous blk.sets
};

// Array to hold the characterising data
trackinfo tracktable[72];
```

```
void tail_insert(requestptr& tail);
/* Precondition: The pointer variable tail points to
   the tail of a linked list.
   Postcondition: A new (empty) node has been added
   at the tail of the list */

void extract_head(requestptr& head);
/* Precondition: The pointer head points to the
   head of a linked list.
   Postcondition: The head of the list is removed. */

void extract_middle(requestptr& discard);
/* Precondition: 'discard' points to the element to be deleted.
   Postcondition: The element pointed to by 'discard' is
   deleted. */

void extract_tail(requestptr& tail);
/* Precondition: The pointer tail points to the
   tail of a linked list.
   Postcondition: The tail of the list is removed. */

long far_no_saw(float start, float stop, int start_tr, int stop_tr);
/* To be used when next access is more than 200 blocks away
   and no turns are neccesary.
   Precondition: start and stop is physical addresses to
   seek from->to respectively. The respective tracks is
   held in start_tr and stop_tr.
   Postcondition: Returns the model-seektime */

long far_saw(float start, float stop, int start_tr, int stop_tr);
/* To be used when next access is more than 200 blocks away
   and turns are neccesary.
   Precondition: start and stop is physical addresses to
   seek from->to respectively. The respective tracks is
   held in start_tr and stop_tr.
   Postcondition: Returns the model-seektime */

long near_same_tr(float start, float stop, int tr);
/* To be used when next access is less than 200 blocks away
   and on same trackset.
   Precondition: start and stop is physical addresses to
   seek from->to respectively. The track is held in tr.
   Postcondition: Returns the model-seektime */

long near_other_tr(float start, float stop, int start_tr, int stop_tr);
/* To be used when next access is less than 200 blocks away
   and turns are neccesary.
```

```
    Precondition: start and stop is physical addresses to
    seek from->to respectively. The respective tracks is
    held in start_tr and stop_tr.
    Postcondition: Returns the model-seektime */

int main(int argc, char *argv[])
{
  // Starting time
  long starttime, stoptime;
  struct timeval time, *tp;
  tp = &time;

  gettimeofday(tp, NULL);
  starttime = ((tp->tv_sec%1000000)*1000)+(tp->tv_usec/1000);

  // General variables
  const char *requestfile = "no_file_specified";
  const char *bottomfile = "no_file_specified";
  char thrash[16];
  int scan_amount = 16;

  // Get in-parameters
  if(argc >= 2)
    requestfile = argv[1];
  if(argc >= 3)
    bottomfile = argv[2];
  if(argc >= 4)
    scan_amount = atol(argv[3])-1;

  // Open the requestfile
  ifstream req_file;
  req_file.open(requestfile);
  if (req_file.fail())
    {
      cout << "Kan ikke aapne req_file <" << requestfile << ">\n";
      exit(1);
    }

  // Open the bottomfile
  ifstream end_of_tracks;
  end_of_tracks.open(bottomfile);
  if (end_of_tracks.fail())
    {
      cout << "Kan ikke aapne end_of_tracks <" << bottomfile << ">\n";
      exit(1);
    }

  // Putting characterising data into tracktable
```

```
end_of_tracks ≫ tracktable[0].akk_N;
end_of_tracks ≫ thrash;       // Second coloumn
tracktable[0].N = tracktable[0].akk_N;

for (int i=1; i≤71; i++)
  {
    end_of_tracks ≫ tracktable[i].akk_N;
    end_of_tracks ≫ thrash;       // Second coloumn
    tracktable[i].N = tracktable[i].akk_N - tracktable[i-1].akk_N;
  }

// Put blocknumbers and readlength from file into request array
// Read the first line (the header)
req_file ≫ thrash;
req_file ≫ thrash;

// First access
requestptr unsorted_head;
requestptr unsorted_tail;

unsorted_head = new request;
unsorted_tail = unsorted_head;

req_file ≫ unsorted_head→addr;
req_file ≫ unsorted_head→readlen;
unsorted_head→prev = NULL;
unsorted_head→next = NULL;

if (unsorted_head→addr < tracktable[0].akk_N)       // Position on 1st trackset
  {
    unsorted_head→pos = unsorted_head→addr;
    unsorted_head→track = 0;
  }
else if (unsorted_head→addr > tracktable[71].akk_N)       // Illegal position
  {
  cout ≪ "Illegal request:  " ≪ unsorted_head→addr ≪ " Exiting\n";
  exit(1);
  }

else      // Repeat until correct trackset is found
  {
  int a = 1;
  while (!((unsorted_head→addr > tracktable[a-1].akk_N) &&
           (unsorted_head→addr ≤ tracktable[a].akk_N)))
    {
      a++;
    }
```

```
    if (a%2 == 0)       // Forward track
      unsorted_head→pos = unsorted_head→addr - tracktable[a-1].akk_N;
    else       // Reverse track
      unsorted_head→pos = tracktable[a].akk_N - unsorted_head→addr;

    unsorted_head→track = a;
    }

  // Rest of the accesses
  for (int i=1; i≤scan_amount; i++)
    {
      tail_insert(unsorted_tail);
      req_file ≫ unsorted_tail→addr;
      req_file ≫ unsorted_tail→readlen;

      // Position on 1st trackset
      if (unsorted_tail→addr < tracktable[0].akk_N)
        {
          unsorted_tail→pos = unsorted_tail→addr;
          unsorted_tail→track = 0;
        }
      else if (unsorted_tail→addr > tracktable[71].akk_N)       // Illegal position
        {
          cout ≪ "Illegal request:  " ≪ unsorted_tail→addr ≪ " Exiting\n";
          exit(1);
        }

      else       // Repeat until correct trackset is found
        {
          int a = 1;
          while (!((unsorted_tail→addr > tracktable[a-1].akk_N) &&
                   (unsorted_tail→addr ≤ tracktable[a].akk_N)))
            {
              a++;
            }
          if (a%2 == 0)       // Forward track
            unsorted_tail→pos = unsorted_tail→addr - tracktable[a-1].akk_N;
          else       //  Reverse track
            unsorted_tail→pos = tracktable[a].akk_N - unsorted_tail→addr;

          unsorted_tail→track = a;
        }
    }

  // Sort the linked list with respect to SLTF model
  // starting from BOT (phys.pos=0)
  long modeltime = 0;
  request last;
```

```
long slt = 240000;
requestptr sltptr;

requestptr sorted_head;
requestptr sorted_tail;
sorted_head = new request;
sorted_tail = sorted_head;

requestptr examine = unsorted_head;

if ((examine→track%2 == 0) && (examine→pos ≤ TOOTH))      // Forward, close
  modeltime = near_other_tr(0, examine→pos, 0, examine→track);
else if ((examine→track%2 == 0) && (examine→pos > TOOTH))      // Forward, far
  modeltime = far_no_saw(0, examine→pos, 0, examine→track);
else
  modeltime = far_saw(0, examine→pos, 0, examine→track);      // Reverse

if (modeltime < slt)
  {
    slt = modeltime;
    sltptr = examine;
  }

if (examine→next ≠ NULL)
  do
    {
      examine = examine→next;
      if ((examine→track%2 == 0)
          && (examine→pos ≤ TOOTH))      //Forward, close
        modeltime = near_other_tr(0, examine→pos, 0, examine→track);
      else if ((examine→track%2 == 0)
                && (examine→pos > TOOTH))      //Forward,far
        modeltime = far_no_saw(0, examine→pos, 0, examine→track);
      else
        modeltime = far_saw(0, examine→pos, 0, examine→track);      // Reverse

      if (modeltime < slt)
        {
          slt = modeltime;
          sltptr = examine;
        }
    } while (examine→next ≠ NULL);

last = *sltptr;      // Keep the latest values

*sorted_head = *sltptr;      // This is the first element in the sorted list
sorted_head→loctime = slt;
```

```
sorted_head→next = NULL;
sorted_head→prev = NULL;

// Remove sltptr
if (sltptr == unsorted_head)
  {
    examine = unsorted_head→next;
    sltptr = unsorted_head→next;
    extract_head(unsorted_head);
  }
else if (sltptr == unsorted_tail)
  {
    examine = unsorted_head;
    sltptr = unsorted_head;
    extract_tail(unsorted_tail);
  }
else
  {
    examine = unsorted_head;
    extract_middle(sltptr);
  }

// Sort the rest of the unsorted_list
if (examine ≠ NULL)
  do
    {
      slt = SLT_INIT;

      if (examine→track == last.track)       // Same track
        if (examine→track%2 == 0)       // Same, forward track
          if (examine→pos > last.pos)       // Same, forward, continue
            if ((examine→pos - last.pos) ≤ TOOTH)       //Same,forward,cont,close
              modeltime = near_same_tr(last.pos, examine→pos, last.track);
            else       // Same, forward, cont, far
              modeltime = far_no_saw(last.pos, examine→pos,
                                     last.track, examine→track);
          else       // Same, forward, back  (close == far)
            modeltime = far_saw(last.pos, examine→pos,
                                last.track, examine→track);
        else       // Same, reverse track
          if (examine→pos < last.pos)       // Same, reverse, continue
            if ((last.pos - examine→pos) ≤ TOOTH)       //Same,reverse,cont,close
              modeltime = near_same_tr(last.pos, examine→pos, last.track);
            else       // Same, reverse, cont ,far
              modeltime = far_no_saw(last.pos, examine→pos,
                                     last.track, examine→track);
          else       // Same, reverse, back (close == far)
            modeltime = far_saw(last.pos, examine→pos,
```

```
                                    last.track, examine→track);

    else      // Other track
      if (examine→track%2 == last.track%2)      // Other, same track direction
        if (examine→track%2 == 0)      // Other, same_tr_dir, forward track
          if (examine→pos > last.pos)      //Other,same_tr_dir,fw,continue
            if ((examine→pos - last.pos) ≤ TOOTH)      //Oth,same,fw,cont,clos
              modeltime = near_other_tr(last.pos, examine→pos,
                                        last.track, examine→track);
            else      // Other, same_tr_dir, forward, continue, far
              modeltime = far_no_saw(last.pos, examine→pos,
                                     last.track, examine→track);
          else      // Other, same_tr_dir, forward, back (close == far)
            modeltime = far_saw(last.pos, examine→pos,
                                last.track, examine→track);
        else      // Other, same_tr_dir, reverse track
          if (examine→pos < last.pos)      //Other,same_tr_dir,rv,continue
            if ((last.pos - examine→pos) ≤ TOOTH)      //Oth,same,rv,cont,clos
              modeltime = near_other_tr(last.pos, examine→pos,
                                        last.track, examine→track);
            else      // Other, same_tr_dir, reverse, continue, far
              modeltime = far_no_saw(last.pos, examine→pos,
                                     last.track, examine→track);
          else      // Other, same_tr_dir, reverse, back (close == far)
            modeltime = far_saw(last.pos, examine→pos,
                                last.track, examine→track);
      else      // Other, different track direction
        if (examine→track%2 == 1)      // Other, diff, examine on reverse track
          if (examine→pos < last.pos)      // Other, diff, e_o_r_t, turn
            if ((last.pos - examine→pos) ≤ TOOTH)      //O,d,e_o_r_t,trn,close
              modeltime = near_other_tr(last.pos, examine→pos,
                                        last.track, examine→track);
            else      // Other, diff, e_o_r_t, turn, far
              modeltime = far_no_saw(last.pos, examine→pos,
                                     last.track, examine→track);
          else      // Other, diff, e_o_r_t, cont (close == far)
            modeltime = far_saw(last.pos, examine→pos,
                                last.track, examine→track);
        else      // Other, diff, examine on forward track
          if (examine→pos > last.pos)      // Other, diff, e_o_f_t, turn
            if ((examine→pos - last.pos) ≤ TOOTH)      //O,d,e_o_f_t,trn,close
              modeltime = near_other_tr(last.pos, examine→pos,
                                        last.track, examine→track);
            else      // Other, diff, e_o_f_t, turn, far
              modeltime = far_no_saw(last.pos, examine→pos,
                                     last.track, examine→track);
          else      // Other, diff, e_o_f_t, cont (close == far)
            modeltime = far_saw(last.pos, examine→pos,
```

```
                                    last.track, examine→track);

if (modeltime < slt)
  {
    slt = modeltime;
    sltptr = examine;
  }

if (examine→next ≠ NULL)
  do      // repeat the same for the rest of the unsorted_list
    {
      examine = examine→next;

      if (examine→track == last.track)       // Same track
        if (examine→track%2 == 0)       // Same, forward track
          if (examine→pos > last.pos)       // Same, forward, continue
            if ((examine→pos - last.pos) ≤ TOOTH)       //S,fw,cont,close
              modeltime = near_same_tr(last.pos, examine→pos,
                                       last.track);
            else      // Same, forward, cont, far
              modeltime = far_no_saw(last.pos, examine→pos,
                                     last.track, examine→track);
          else      // Same, forward, back  (close == far)
            modeltime = far_saw(last.pos, examine→pos,
                                last.track, examine→track);
        else      // Same, reverse track
          if (examine→pos < last.pos)       // Same, reverse, continue
            if ((last.pos - examine→pos) ≤ TOOTH)       //S,rev,cont,close
              modeltime = near_same_tr(last.pos, examine→pos,
                                       last.track);
            else      // Same, reverse, cont ,far
              modeltime = far_no_saw(last.pos, examine→pos,
                                     last.track, examine→track);
          else      // Same, reverse, back (close == far)
            modeltime = far_saw(last.pos, examine→pos,
                                last.track, examine→track);

      else      // Other track
        if (examine→track%2 == last.track%2)       // Other, same_tr_dir
          if (examine→track%2 == 0)       // Other, same_tr_dir, fw track
            if (examine→pos > last.pos)       //Other,same_tr_dir,fw,cont
              if ((examine→pos - last.pos) ≤ TOOTH)       //O,s,fw,c,clos
                modeltime = near_other_tr(last.pos, examine→pos,
                                          last.track, examine→track);
              else      // Other, same_tr_dir, forward, continue, far
                modeltime = far_no_saw(last.pos, examine→pos,
                                       last.track, examine→track);
            else      // Other, same_tr_dir, forward, back (close == far)
```

```
                modeltime = far_saw(last.pos, examine→pos,
                                    last.track, examine→track);
            else      // Other, same_tr_dir, reverse track
              if (examine→pos < last.pos)      //Other,same_tr_dir,rv,cont
                if ((last.pos - examine→pos) ≤ TOOTH)      //O,s,rv,c,clos
                  modeltime = near_other_tr(last.pos, examine→pos,
                                            last.track, examine→track);
                else      // Other, same_tr_dir, reverse, continue, far
                  modeltime = far_no_saw(last.pos, examine→pos,
                                         last.track, examine→track);
              else      // Other, same_tr_dir, reverse, back (close == far)
                modeltime = far_saw(last.pos, examine→pos,
                                    last.track, examine→track);
          else      // Other, different track direction
            if (examine→track%2 == 1)      //O, diff, examine on rev track
              if (examine→pos < last.pos)      // Other, diff, e_o_r_t, turn
                if ((last.pos - examine→pos) ≤ TOOTH)      //O,d,e,t,close
                  modeltime = near_other_tr(last.pos, examine→pos,
                                            last.track, examine→track);
                else      // Other, diff, e_o_r_t, turn, far
                  modeltime = far_no_saw(last.pos, examine→pos,
                                         last.track, examine→track);
              else      // Other, diff, e_o_r_t, cont (close == far)
                modeltime = far_saw(last.pos, examine→pos,
                                    last.track, examine→track);
            else      // Other, diff, examine on forward track
              if (examine→pos > last.pos)      // Other, diff, e_o_f_t, turn
                if ((examine→pos - last.pos) ≤ TOOTH)      //O,d,e,t,close
                  modeltime = near_other_tr(last.pos, examine→pos,
                                            last.track, examine→track);
                else      // Other, diff, e_o_f_t, turn, far
                  modeltime = far_no_saw(last.pos, examine→pos,
                                         last.track, examine→track);
              else      // Other, diff, e_o_f_t, cont (close == far)
                modeltime = far_saw(last.pos, examine→pos,
                                    last.track, examine→track);

        if (modeltime < slt)
          {
            slt = modeltime;
            sltptr = examine;
          }

      } while (examine→next ≠ NULL);

last = *sltptr;      // Keep the latest values
tail_insert(sorted_tail);
sorted_tail→addr = sltptr→addr;
```

```
        sorted_tail→readlen = sltptr→readlen;
        sorted_tail→pos = sltptr→pos;
        sorted_tail→track = sltptr→track;
        sorted_tail→loctime = slt;

        // Remove sltptr
        if (sltptr == unsorted_head)
          {
            examine = unsorted_head→next;
            sltptr = unsorted_head→next;
            extract_head(unsorted_head);
          }
        else if (sltptr == unsorted_tail)
          {
            examine = unsorted_head;
            sltptr = unsorted_head;
            extract_tail(unsorted_tail);
          }
        else
          {
            examine = unsorted_head;
            extract_middle(sltptr);
          }

      } while (unsorted_head ≠ NULL);

// Get the time and calculate the time spent
gettimeofday(tp, NULL);
stoptime = ((tp→tv_sec%1000000)*1000)+(tp→tv_usec/1000);
stoptime = stoptime - starttime;

cout ≪ "Time_used:   " ≪ stoptime ≪ endl;

// Write out the sorted list
requestptr outptr = sorted_head;
if (outptr == NULL)
  cout ≪ "Empty list\n";
else
  {
    cout ≪ outptr→addr ≪ "  " ≪ outptr→readlen
         ≪ "  " ≪ outptr→pos ≪ "  "
         ≪ outptr→track ≪ "  " ≪ outptr→loctime ≪ endl;      // DEBUG!!!
    do
      {
        outptr = outptr→next;
        cout ≪ outptr→addr ≪ "  " ≪ outptr→readlen
             ≪ "  " ≪ outptr→pos ≪ "  "
             ≪ outptr→track ≪ "  " ≪ outptr→loctime ≪ endl;      // DEBUG!!!
```

```
        } while (outptr→next ≠ NULL);
    }

  // Cleanup
  req_file.close();
  end_of_tracks.close();
}  // End of main()

/*****************************************/

void tail_insert(requestptr& tail)
{
  requestptr temp_ptr;
  temp_ptr = new request;
  if (temp_ptr == NULL)
    {
      cout ≪ "Error:  Insufficient storage.  Exiting...\n";
      exit(1);
    }
  temp_ptr→next = NULL;
  temp_ptr→prev = tail;
  tail→next = temp_ptr;
  tail = temp_ptr;
}

void extract_head(requestptr& head)
{
  requestptr temp_ptr;
  temp_ptr = head;
if (head→next == NULL)
  {
    head = NULL;
    delete temp_ptr;
  }
else
  {
    head = head→next;
    head→prev = NULL;
    delete temp_ptr;
  }
}

void extract_middle(requestptr& discard)
{
  requestptr temp_ptr;
  temp_ptr = discard;
  discard = discard→prev;
  temp_ptr→prev→next = temp_ptr→next;
```

```
  temp_ptr→next→prev = temp_ptr→prev;
  delete temp_ptr;
}

void extract_tail(requestptr& tail)
{
  requestptr temp_ptr;
  temp_ptr = tail;
if (tail→prev == NULL)
  {
    tail = NULL;
    delete temp_ptr;
  }
else
  {
    tail = tail→prev;
    tail→next = NULL;
    delete temp_ptr;
  }
}

long far_no_saw(float start, float stop, int start_tr, int stop_tr)
{
  float temp;
  temp = T_0 + T_WIND
    * fabs(stop / tracktable[stop_tr].N - start / tracktable[start_tr].N);
  return (long)floor(temp + 0.5);
}

long far_saw(float start, float stop, int start_tr, int stop_tr)
{
  float temp;
  temp = T_0 + T_WIND
    * fabs(stop / tracktable[stop_tr].N - start / tracktable[start_tr].N)
    + T_REW + T_SAW;
  return (long)floor(temp + 0.5);
}

long near_same_tr(float start, float stop, int tr)
{
  float temp;
  temp = T_WIND / tracktable[tr].N * fabs(stop - start);
  return (long)floor(temp + 0.5);
}

long near_other_tr(float start, float stop, int start_tr, int stop_tr)
{
  float temp;
```

```
  temp = T_0 + T_WIND
    * fabs(stop / tracktable[stop_tr].N - start / tracktable[start_tr].N)
    + T_REW + T_SAW;
  return (long)floor(temp + 0.5);
}
```

```
//File Name: MLR1driver.cc
//Author: Thomas Maukon Andersen
/*Description:  This program takes as input a list of
 blocknumbers, readlengths and physical positions.
 It then seeks and reads the tape in the order specified
 in the input-file. Startposition, stopposition,
 difference and time of the locate operation together with
 the readlength and the readtime is written to stdout.
 The total time theese seeks took is also written to stdout.*/
//Last Changed: November 8, 1997
//--------------------------------

#include "../station/streamer/tapestreamer.h"
#include <stdio.h>
#include <math.h>
#include <iostream.h>
#include <stdlib.h>
#include <fstream.h>
#include <time.h>

main(int argc, char *argv[])
{
  long seektime = 0, readtime = 0;
  long seek_from, seek_to, difference, readlength, readcounter;
  const char *filename = "no_file_specified";
  char thrash[16];
  int phys_posi;

  // Get in-parameters
  if(argc >= 2)
        filename = argv[1];

  // Mount tapestation
  const char *tapename = getenv("TAPE");
  if(tapename == NULL)
        tapename = "/dev/rmt/0hn";

  // Create tapestreamer-object
  TapeStreamer *streamer;
  streamer = new TapeStreamer(tapename);

  // Rewind and disable compression
  streamer->set_no_compression();
  streamer->rewind();

  // Space for data
  unsigned char data[BLOCKSIZE];
```

```
// The time
time_t starttime, stoptime;
long msecs, oldmsecs;
struct timeval time_, *tp;
tp = &time_;

// The position
long posi, *pos = &posi;

// Return value of streamer function calls
int ret = 0;

// Open the datafile
ifstream innfil;
innfil.open(filename);
if (innfil.fail())
        {
          cout << "Kan ikke aapne innfil <" << filename << ">\n";
          exit(1);
        }

// Read the first line (the header)
innfil >> thrash;
innfil >> thrash;

// Get the first position
innfil >> posi;
innfil >> readlength;
innfil >> phys_posi;
seek_from = 0;      //starting from BOT
seek_to = posi;

// Get starttime for characterizing process
starttime = time(NULL);

// Write a header
cout << "#Phys_pos Seek_from Seek_to Difference "
     << "SeekTime ReadLength ReadTime\n";

while((! innfil.eof()) && (ret >= 0))
        // Continue until datafile is empty or an error occurs
        {

        // Get time
        gettimeofday(tp, NULL);
        oldmsecs = ((tp->tv_sec%1000000)*1000) + (tp->tv_usec/1000);

        // Locate next block
```

```
        ret = streamer→goto_posi(pos, LOGISK);

        // Get time and calculate difference
        gettimeofday(tp, NULL);
        msecs = ((tp→tv_sec%1000000)*1000) + (tp→tv_usec/1000);
        seektime = msecs - oldmsecs;
        oldmsecs = msecs;

        // Read block(s)
        readcounter = 1;
        while ((ret ≥ 0) && (readcounter ≤ readlength))
          {
            ret = streamer→read_block((char *)data);
            readcounter++;
          }

        // Get time and calculate read time
        gettimeofday(tp, NULL);
        msecs = ((tp→tv_sec%1000000)*1000) + (tp→tv_usec/1000);
        readtime = msecs - oldmsecs;

        // Calculate the seek-difference
        difference = seek_to - seek_from;

        // Write out the result
        cout ≪ phys_posi ≪ " " ≪ seek_from
             ≪ " " ≪ seek_to ≪ " "
             ≪ difference ≪ " " ≪ seektime ≪ " "
             ≪ readlength ≪ " " ≪ readtime ≪ endl;

        // Get next position and readlength
        seek_from = seek_to + readlength;
        innfil ≫ posi;
        innfil ≫ readlength;
        innfil ≫ phys_posi;
        seek_to = posi;
        }

if(ret<0)       // An error must have occured
        cout ≪ "Ret=" ≪ ret ≪ " Program encountered an error and died!\n";

// Get stoptime and calculate time spent
stoptime = time(NULL);
stoptime = stoptime - starttime;

cout ≪ "#Total time used (s):  "
                ≪ stoptime ≪ endl;
cout ≪ "#Which is " ≪ (stoptime/60) ≪ " mins and "
```

```
        ≪ (stoptime%60) ≪ "secs\n";
  cout ≪ "#Or " ≪ (stoptime/3600.0) ≪ " hours.\n";

  // Clean up!
  delete streamer;
  innfil.close();
}
```