```
//***************************************************************************
// SensorDevices.cc Messtatioen
// Author:          M. Thaler
// Date:      3/2011
//***************************************************************************

//***************************************************************************
// system includes

#include <unistd.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>

#include <sys/socket.h>
#include <netdb.h>
#include <string.h>

//***************************************************************************
// local includes

#include "defs.h"

//***************************************************************************
// local constants

#define MAX_ITERATIONS  100000
#define THE_OVERLAP     2

//***************************************************************************
// local data

int overlap = THE_OVERLAP;
int globalK = 0;

//***************************************************************************
// local procedures

int connToServer(char *hostname, int port);

void SignalHandler(int sig) {
  printf("\nSensors process receiving termination signal\n");
  globalK = MAX_ITERATIONS + 1;
}

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

int numOfSensors(int num) {
  static int numOfs = 0;
  if (num > numOfs) {
    if (num <= SENSOR_MAX_NUM)
      numOfs = num;
    else
      numOfs = -1;
  }
  return(numOfs);
}

//************************************************************************
// equally distributed random values in the range low ... high

int intRand(int low, int high) {
  int     lp, res;
  double  lo, hi, dif, ran;

  if (low <= 0)
        lp = (-1)*low + 1;
  else
        lp = 0;
  lo  = low + lp - 0.5;
  hi  = high + lp + 0.499;    // make sure not to round too much
  dif = hi - lo;
  ran = random();
  ran = lo + dif * ran/RAND_MAX;
  res = (int)(ran + 0.5);
  res -= lp;
  if (res > high)
        res = high;                   // make sure not to round too much
  if (res < low)
            res = low;
  return res;
}

//*****************************************************************************
// generate sequence of temp devices

void RandomSequence(int *seq, int number) {
  static int reservation[2*SENSOR_MAX_NUM];
    static int firstRun = 1;
    int devCount[SENSOR_MAX_NUM];

  int idx, i, tmp;

  if (number > numOfSensors(0)) {
    printf("sequence: too many devices\n");
    exit(0);
  }

    if (firstRun == 1) {
        firstRun = 0;
        for (i = 0; i < numOfSensors(0); i++)
            devCount[i] = 0;
        idx = 0;
      while (idx < number) {
        i = intRand(0, number-1);
        if (devCount[i] < overlap) {
                devCount[i]++;
          reservation[idx] = i;
          idx++;
        }
      }
    }

  for (i = 0; i < number; i++)
        devCount[i] = 0;

    for (i = 0; i < number; i++) {
        reservation[i+number] = reservation[i];
        devCount[reservation[i]]++;
    }

    idx = 0;
  while (idx < number) {
    i = intRand(0, number-1);
    if (devCount[i] < 2) {                 // if not yet twice in list
            if (devCount[i] < 1) {           //    if not in list
                devCount[i]++;
          reservation[idx] = i;
          idx++;
            }
            else {
                tmp = intRand(0, 9);         //    if in list
                if (tmp > 6) {               //        do only for 5%
                    devCount[i]++;
              reservation[idx] = i;
              idx++;
                }
            }
    }
  }

    for (i = 0; i < number; i++) {
      *seq = reservation[i+number];
      seq++;
    }
}


//**************************************************************************
// Function: main(), parameter: hostname or IP address in dot format
//**************************************************************************

int main(int argc, char *argv[]) {

  struct sigaction sig;

    int     StationSeq[SENSOR_MAX_NUM];

    int     sfd, maxWait, i, j, rand;
  int     anzSensors;
    char    buf[BUF_SIZE];
    SensorData  sensor;
    float   deltaT;

    float       tempPreset[8]  = {20, 45, 30, 20, 15, 10, 15, 20};
    float       startup[8]     = {0, 0, 0, 0, 0, 0, 0, 0};
    int         sequenceNr[8]  = {0, 0, 0, 0, 0, 0, 0, 0};

    //*** check for hostname ... a kind of hack

    if (argc < 4)  {
        printf("Need number of devices, hostname or IP address and port number\n");
        exit(-1);
    }

  if ((anzSensors = numOfSensors(atoi(argv[1]))) < 0) {
    printf("\n*** invalid number of sensor devices ***\n\n");
    exit(0);
  }

  // set up signal handlers
  sigemptyset(&sig.sa_mask);
  sig.sa_handler = SignalHandler;
  sig.sa_flags = 0;
  sigaction(SIGTERM, &sig, NULL);
  sigaction(SIGKILL, &sig, NULL);
  sigaction(SIGINT,  &sig, NULL);

    sleep(2);
  printf("Sensor device starting up\n");

    globalK = 0;
    while (globalK < MAX_ITERATIONS) {

      RandomSequence(StationSeq, anzSensors);

      for (i = 0; i <  anzSensors; i++) {   // for all devices
            deltaT = intRand(-2, 2);
            sensor.deviceID   = StationSeq[i];
            sensor.sequenceNr = sequenceNr[sensor.deviceID];
            sensor.valIS      = deltaT + startup[sensor.deviceID];
            sensor.valREF     = tempPreset[sensor.deviceID];
      sensor.status     = 0;

            sequenceNr[sensor.deviceID]++;
            sfd = connToServer(argv[2], atoi(argv[3]));
        write(sfd, (char *)&sensor,sizeof(SensorData));
        close(sfd);
        maxWait   = 4000000;
        maxWait   = maxWait / anzSensors;
        rand     = intRand(maxWait/3, maxWait);
        usleep(rand);
        }
        for (j = 0; j < anzSensors; j++) {
            if (startup[j] < tempPreset[j])
                startup[j] += 2;
            else
                startup[j] = tempPreset[j];
        }
      globalK++;
    }
    exit(0);

} // end main

//********************************************************************************
// socket client

int connToServer(char *hostname, int port) {
    int  sfd, sysRet;
    char stringPort[8];
    struct addrinfo hints, *aiList, *aiPtr = NULL;

    sprintf(stringPort, "%d", port);

    memset(&hints, '\0', sizeof(hints));
    hints.ai_flags    = AI_CANONNAME;
    hints.ai_family   = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;

    sysRet = getaddrinfo(hostname, stringPort, &hints, &aiList);
    if (sysRet != 0) {
        printf("error getting network address %s\n", gai_strerror(sysRet));
        return(-1);
    }

    aiPtr = aiList;
    while (aiPtr != 0) {
        sfd = socket(aiPtr->ai_family, aiPtr->ai_socktype, aiPtr->ai_protocol);
        if (sfd >= 0) {
            sysRet = connect(sfd, aiPtr->ai_addr, aiPtr->ai_addrlen);
            if (sysRet == 0)
                break;
            else
                close(sfd);
        }
        aiPtr = aiPtr->ai_next;
    }
    if (aiPtr == NULL) {
        return(-1);
    }
    else
        return(sfd);
}

//********************************************************************************
```