Protecta C++ algorithms test

V8

External (online) help can be used, but not for the algorithm itself. English - hungarian translator must not be used!

1. Two files (A and B) are given. Check if file B starts with the same bytes as the whole file A (can return true if B is bigger) using checksum calculation.
   - Assume a predefined function is available for checksum calculation: uint32_t MyChkSum(const char* buff, size_t len, uint32_t prevchk)
   - MyChkSum can calculate only 1024 bytes (len <= 1024), must be called multiple times for larger files
   - prevchk must set to previous result of MyChkSum, or to 0 for the first run
   - B can be much bigger (e.g.: check header in a several GB media file), avoid reading the whole B if not necessary.
   - **Implement the following function:**
     bool Compare(const std::string& p_A_filename, const std::string& p_B_filename) returns true if B starts with the same bytes as the whole A file

2. A set of points on a plane is given. Each point is identified by its x and y coordinates and a text identifier (id). More than one point can share the same identifier (coordinates can be different).
   - **implement the following function:**
     float CalculateBiggestRadius(const std::vector<Point>& points) - the function should return the radius of the largest circle centred at origin (x=0,y=0) in which the identifier of every point is unique. A point is inside the circle if the distance from origin is less than the radius. Write the algorithm in C++!

3. Implement the server side of the communication handling of a client-server communication in C++.

A communication medium is given (represented by a singleton of class Communication in the server, COMM) through which multiple clients can communicate with a central server. The communication is always initiated by the client.

- COMM object assigns a unique id to every client (can be reused but no two active clients have the same id).
- The COMM object can identify the client from the message implicitly

The following types of messages exist:
START - client should always start the communication with that message
STOP   - client should always close the communication with that message

KEEP - dummy message from client (to keep the connection alive)
TEXT - client sends a character string (256 bytes max.) using this type of message to the server
ABORT - sent by the server with error cause in case of communication error
BUSY - sent by the server if too many simultaneous connections exist

The client should send at least one message in every 30 secs (IDLE_TIMEOUT) (if no text available, it should send a KEEP message)

Beside the type and other fields mentioned above the message has the following additional properties:

- sequence number - sent by the client and incremented in every message (started from 1).

The Communication class has the following methods:

- GetMessage - waits no more than the timeout (millisec) given as a parameter and returns a message in a dynamically allocated buffer if available
- SendMessage - Sends a message to the client (id in the message) (ABORT or BUSY)

The following convenience function is also available:

- unsigned long msElapsed() - returns the time elapsed (since program start) in millisecs (can overflow!)

Write the connection handling part of the server:

- **Define the interface of the Communication class** (GetMessage, SendMessage, message struct) (Not implementation, just declaration of GetMessage, SendMessage)
- **Implement the message handling loop**
    - read messages from clients (Communication::GetMessage)
    - check correct message order from the client, START, …., STOP (Send ABORT on error, Communication::SendMessage)
    - check sequence number (ABORT if not increased by one)
    - ABORT if no message from a client for more than IDLE_TIMEOUT
    - print TEXT message to the console
    - No more than MAX_CONN active connection can exist at the same time. (Reply BUSY to clients above the limit)