Video filtering

Submission deadline:	**2021-03-21 23:59:59**	Zbývá **4 dní, 21h, 34m a 36s**
Late submission with malus:	**2021-06-30 23:59:59** (Late submission malus: 100.0000 %)	
Evaluation:	**0.0000**	
Max. assessment:	**5.0000** (Without bonus points)	
Submissions:	0 / 20 Free retries + 20 Penalized retries (-2 % penalty each retry)	
Advices:	0 / 2 Advices for free + 2 Advices with a penalty (-10 % penalty each advice)	

The task is to develop a function that filters unwanted audio streams from binary video file.

The problem works with binary files that represent video data. The format is inspired by standard mkv file format, however, it is significantly simplified for the purpose of this homework. Moreover, the homework is divided into mandatory, optional and bonus tests.

A video file shall store both image stream and audio stream(s) information. We assume that the video contains exactly one image stream, however, there might be more audio streams in the video filer (such as language variants). Since the video file is often streamed, the images and audio streams must be physically interleaved in the file. An example is per-frame interleaving, where there is one image followed by a short portion the audio stream (or a short portion of all available audio streams if there is more than one audio stream present).

The required function is passes three parameters: the file name of the source video file, the file name of the target video file and a language name (an abbreviation of, such as cs, en, fr, ...). The function shall copy the contents of the source video file into the target file. It shall preserve the video stream, however, it shall remove all audio streams except the audio stream in the third parameter. The interface is:

```
#ifndef __PROGTEST__
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <cstdint>
#include <cassert>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <string>
#include <vector>
using namespace std;

const uint32_t BIG_ENDIAN_ID    = 0x4d6d6d4d;
const uint32_t LITTLE_ENDIAN_ID = 0x49696949;
const uint32_t FRAME_ID         = 0x46727246;
const uint32_t VIDEO_DATA_ID    = 0x56696956;
const uint32_t AUDIO_LIST_ID    = 0x416c6c41;
const uint32_t AUDIO_DATA_ID    = 0x41757541;
#endif /* __PROGTEST__ */

bool  filterFile          ( const char      * srcFileName,
                            const char      * dstFileName,
                            const char      * lang );
```

The function shall return `true` for success or `false` for a failure. A failure might be caused by some I/O error (file does not exist/cannot be read/written, ...), or the input file has an invalid format (invalid headers, invalid lengths, ...).

The format of the "video file" is custom, there is some inspiration in the standard mkv file format. The file can be understood as a "binary XML". There are sections that are described by a header (identifier, some attributes, length of its payload). Each section may hold nested section or some unstructured binary data in its payload. For instance a global section holds individual frames in its payload. On the other hand, VideoData section only holds bytes that form the encoded image. A simplified format of the sections is depicted below:

```
Global section
  Frame
    VideoData
    AudioList
      AudioData
      AudioData
      ...
  Frame
    VideoData
    AudioList
      AudioData
      AudioData
      ...
  ...
```

Global header
: It is always present at the beginning of the file. It requires 12 + 4 B, the structure is:

```
offset        type         meaning
+0            uint32       global header ID: 0x49696949 or 0x4d6d6d4d
+4            uint32       number of frames in the payload
+8            uint32       length of the payload
+12 payload (Frames)
+12 + length  uint32       CRC32
```

Global header ID is either 0x49696949 or 0x4d6d6d4d. The first value is used if little endian is used, the second means big endian. More on file endianity below.

The number of frames simply describes the total # of nested Frame sections in the payload.

Length is the total number of bytes in the payload (i.e. sum of bytes in all nested sections, including the headers. There is a final uint32 value appended after the payload. The value is a checksum (CRC32). The checksum may be used to validate the contents of the header and the payload, it may detect unwanted modifications (e.g. I/O errors). The checksum is optional, see below.

Frame
: represents one video frame. The structure is:

```
offset        type         meaning
+0            uint32       ID Frame: 0x46727246
+4            uint32       length of the payload
+8 payload (VideoData a AudioList)
+8 + length   uint32       CRC32
```

One frame consist of two subsections: VideoData and AudioList. Both subsections must be present, each exactly once. The first nested subsection must be VideoData, the second AudioList. The other fields (length, CRC32) are similar to that in Global header.

VideoData
: represents the bytes that code the image of the frame. The structure is:

```
offset        type         meaning
+0            uint32       ID VideoData: 0x56696956
+4            uint32       length of the payload
+8 payload (unstructured bytes coding the image)
+8 + length   uint32       CRC32
```

The bytes that code the image are not important. For the purpose of the format, they are just bytes that must be copied without any modifications.

AudioList
: represents the list of available audio streams. The structure is:

```
offset        type         meaning
+0            uint32       ID AudioList: 0x416c6c41
+4            uint32       number of audio streams
+8            uint32       length of the payload
+12 payload (AudioData)
+12 + length  uint32       CRC32
```

The payload consists of AudioData sections. The number of nested AudioData sections is equal to the number of audio streams field in the header.

AudioData
: represents the bytes that code a potion of an audio stream. The structure is:

```
offset        type         meaning
+0            uint32       ID AudioData: 0x41757541
+4            2 chars      language (abbreviation)
+6            uint32       length of the payload
+10 payload (unstructured bytes that code the audio stream)
+10 + length  uint32       CRC32
```

The language field in the header is the language code of the particular audio stream. The language is coded as a pair of characters, e.g. cs for Czech, en for English, ... The payload is formed by unstructured bytes that code the audio stream. Similarly to the VideoData the bytes are to be copied without any modifications if the particular audio stream is used. If the audio stream is not used, then the corresponding audio data and the corresponding AudioData header is omitted.

The filtering of the file means the original is left unmodified. The new file is a copy of the original where the video and selected audio stream is preserved and the other audio streams are omitted. Naturally, the modification changes the size of the file, thus it requires changes to the headers, where the payload size has changed (i.e. AudioList, Frame and Global header).

The homework is divided into mandatory, optional and bonus parts. Mandatory is a correct manipulation of the input file in little-endian coding. Since the running platform is little-endian, this is a simplification since there is no need to modify the byte order while reading/writing the file. Moreover, the mandatory solution does not have to manipulate the checksums. For the mandatory tests, it is OK to fill the CRC32 with zeroes.

The optional part requires correct handling of both little and big endian input files. In the case of big-endian, the program must convert the multibyte values when reading/writing them. In our file format, there are unsigned 32 bit integers affected by the little/big endian conversion. The conversion does not affect the payload of AudioData/VideoData (the payload is considered a stream of bytes). The filtering shall preserve endianity, i.e. if the input is little-endian file, the output shall be little endian, a big endian input file shall be converted into a big endian output file.

If your program correctly implements conversion of little endian files and does not handle big endian files, the program will be awarded some points, however, there will be some penalty for the missing big endian functionality.

The bonus part requires correct handling of the CRC32 checksums. The computation of CRC32 follows the standard CRC32 algorithm with polynomial of 0x04c11db7. When implementing the CRC32 functionality, it is important to design the structure of the program to easily propagate the written bytes into all affected sections. For instance, a byte in the VideoData payload affects the CRC32 values of all parental sections, i.e. VideoData, Frame and Global header. A smart solution is to compute the parental CRCs as a combination of the child CRC32s. The algorithm is described in article **Andrew Kadatch and Bob Jenkins: Everything we know about CRC but afraid to forget**.

The solution that computes CRC32 is significantly longer than the basic solutions. Reference solution without CRC32 is about 120 source lines, the version with CRC32 is about three times longer. Therefore, the CRC32 is left as a bonus for interested students (and for extra points).

The attached archive contains a set of test input / expected output files. The corresponding pairs have the same numbers. The language used to filter the input file is included in the output filename. There is no output file for input #6. Input #6 is invalid, the length in one of the headers does not match the actual size.

A note on valid and invalid input files (the specific cases are mostly included in the attached archive):

- The number of Frames in a file may be arbitrary. Even zero frames is legal, the result is a file with zero frames in this case.
- There does not have to be audio at all in the file. Such file will have AudioList, where the number of streams is 0.
- Different Frames may contain AudioData with different languages. The filtering shall remove all AudioData except the language selected. Thus, it is ok if some Frame(s) in the source do not contain the AudioData in the filtered labguage - just copy the required AudioData in the frames where the selected AudioData exist.
- If the filtered language is e.g. "en" and some (or all) frames do not contain the "en" AudioData, the result will be AudioList with 0 streams (in Frames, where the filtered AudioData were not present), or AudioList with 1 AudioData (where the filtered language is present).
- You do not have to worry about cases, where there are AudioData with the same language twice (or more times) in a single Frame. Such input is not tested.
- Audio/video data may be really long. Do not read the entire file/frame/... into memory, such approach exceeds thememory limit. Copy the data on-the-fly.

Update 2021-03-06: A solution of this problem may be used for code review if it passes all mandatory and optional tests with 100 % results. Moreover, use C++ features (classes, ...) in your solution if you plan to submit it for code review. If you do not know how to use classes yet or if you do not use classes in your implementation, do NOT submit the solution for code review (submit some other homework).

Sample data: Download

Submit: Choose File No file selected Submit

Reference

- **Evaluator: computer**
  - Program compiled
  - **0** Test 'Zakladni test se soubory dle ukazky': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 0.003 s (limit: 10.000 s)
    - Peak mem usage: 6360 KiB (limit: 14789 KiB)
    - Mandatory test success, evaluation: 100.00 %
  - **0** Test 'Nespravne vstupy': success
    - result: 100.00 %, required: 50.00 %
    - Total run time: 0.003 s (limit: 9.997 s)
    - Peak mem usage: 6360 KiB (limit: 14789 KiB)
    - Mandatory test success, evaluation: 100.00 %
  - **0** Test 'Nahodny test': success
    - result: 100.00 %, required: 50.00 %
    - Total run time: 3.246 s (limit: 9.994 s)
    - Peak mem usage: 6360 KiB (limit: 14789 KiB)
    - Mandatory test success, evaluation: 100.00 %
  - **0** Test 'Test osetreni I/O chyb': success
    - result: 100.00 %, required: 50.00 %
    - Total run time: 0.077 s (limit: 6.748 s)
    - Peak mem usage: 6360 KiB (limit: 14789 KiB)
    - Mandatory test success, evaluation: 100.00 %
  - **0** Test 'Nahodny test (big endian)': success
    - result: 100.00 %, required: 70.00 %
    - Total run time: 3.332 s (limit: 15.000 s)
    - Optional test success, evaluation: 100.00 %
  - **0** Test 'Nahodny test (kontrola CRC)': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 6.168 s (limit: 15.000 s)
    - Bonus test - success, evaluation: 130.00 %
  - Overall ratio: 130.00 % (= 1.00 * 1.00 * 1.00 * 1.00 * 1.00 * 1.30)
- Total percent: 130.00 %
- Early submission bonus: 0.50
- Total points: 1.30 * (5.00 + 0.50) = 7.15

SW metrics:

	Total	Average	Maximum	Function name
Functions:	**20**	--	--	--
Lines of code:	**382**	19.10 ± 13.99	55	`filterFile`
Cyclomatic complexity:	**116**	5.80 ± 5.06	19	`filterFile`