СТВОРЕННЯ ТА ОБРОБКА АУДІО ФАЙЛІВ WAVE ФОРМАТУ

Виконала: Биковець Валерія

ПОСТАНОВКА ЗАДАЧІ:

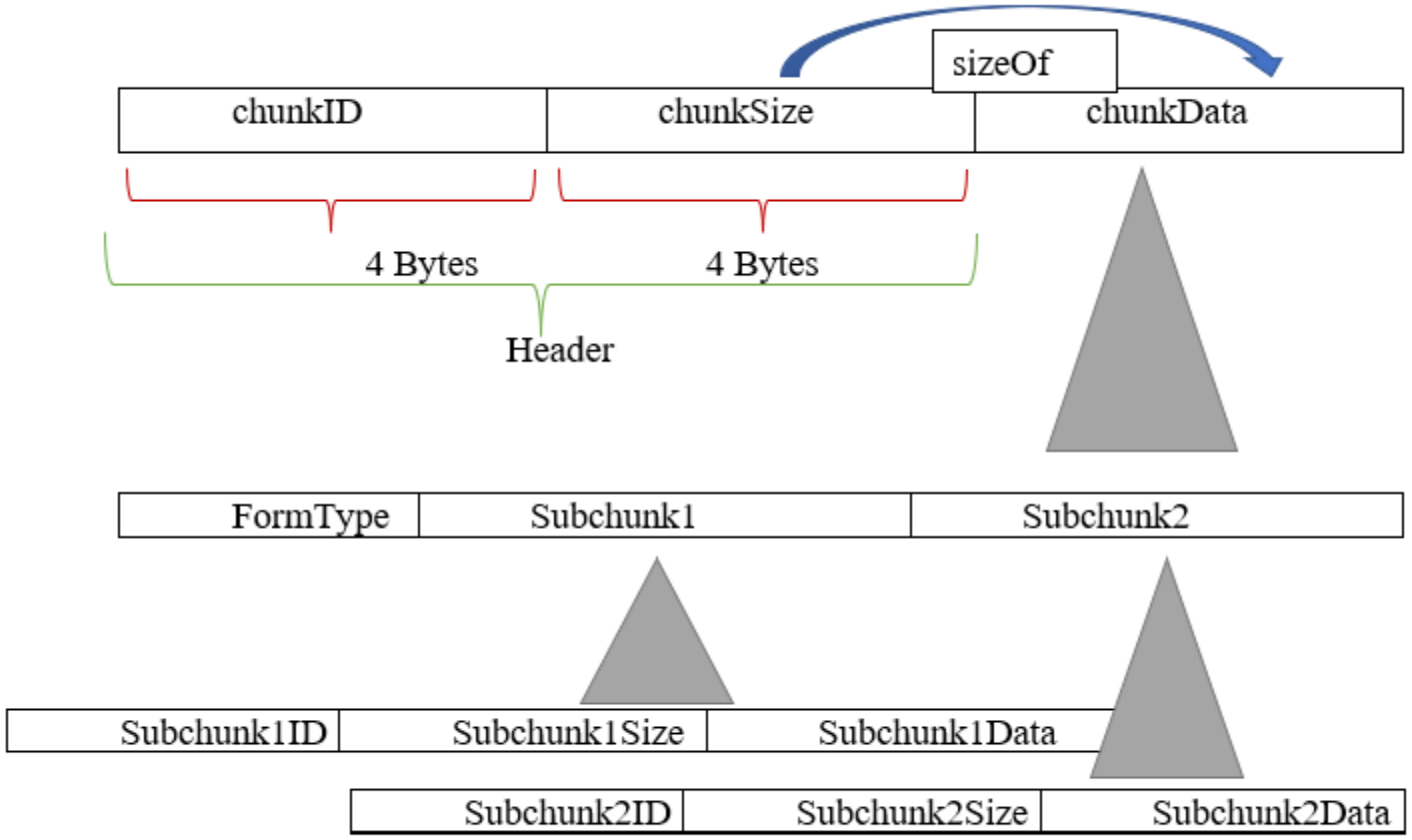
1. ДОСЛІДИТИ АСПЕКТИ СТВОРЕННЯ АУДІО ФАЙЛІВ ФОРМАТУ WAVE

2. ЗРОБИТИ ОГЛЯД МЕТОДІВ ОБРОБКИ ТА ВИЗНАЧИТИ ОСОБЛИВОСТІ ДЛЯ ПРОГРАМНОЇ ОБРОБКИ АУДІО WAVE ФОРМАТУ

3. РОЗРОБИТИ КЛАС МОВОЮ C++, ЯКИЙ БУДЕ ЗРУЧНИМ ІНСТРУМЕНТОМ ДЛЯ РОБОТИ ЗІ СТВОРЕННЯ ТА ОБРОБКИ АУДІО ФОРМАТУ WAVE

4. ПРОДЕМОНСТРУВАТИ ПРИКЛАДИ СТВОРЕННЯ ТА ОБРОБКИ АУДІО ЗА ДОПОМОГОЮ ДАНОГО КЛАСУ

I. RIFF STRUCTURE



Зображення 1. RIFF структура

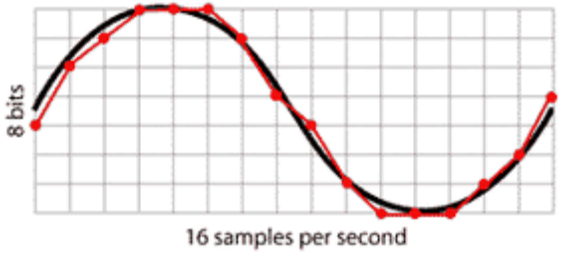
Конкретизація для WAVE

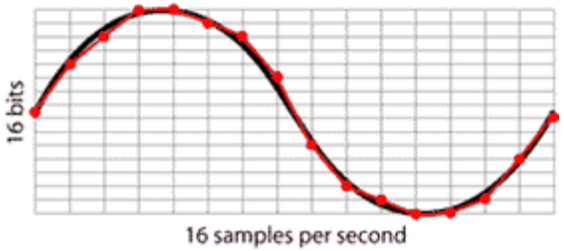
1. Опис RIFF секції: chunkID, chunkSize. Наступні пункти входять до chunkData
2. Format
3. Підсекція fmt : Subchunk1 ID, Subchunk1 Size, chunkData (підсекції fmt): AudioFormat, NumChannels, SampleRate, ByteRate, BlockAlign, BitsPerSample, cbSize
4. Підсекція data: Subchunk2 ID, Subchunk2 Size, data

II. WAVEFORMATEX STRUCTURE

1. Audio Format
2. Sample Rate
3. Number of Channels

4. Bits Per Sample





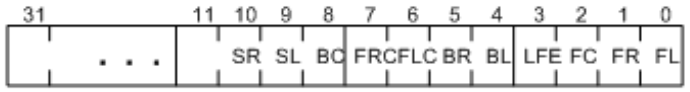
4. ByteRate

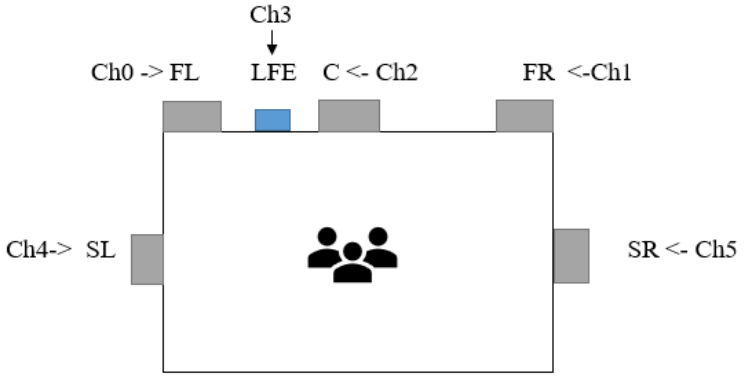
5. Block Align

6. cbSize

WAVEFORMATEXTENSIBLE

Channel Mask





SL – Side Left, FL – Front Left, C – Center, FR – Front Right, SR – Side Right, LFE – Low Frequency

ПОРІВНЯННЯ ЗАСОБІВ

Робота з wave аудіо в Python

```
def write_wave_file(file_name, data):
    with wave.open(file_name, 'wb') as file:
        file.setparams((num_channels, sampwidth, sample_rate, num_samples, 'NONE',
'not compressed'))
        file.writeframes(data)
```

ФУНКЦІОНАЛ

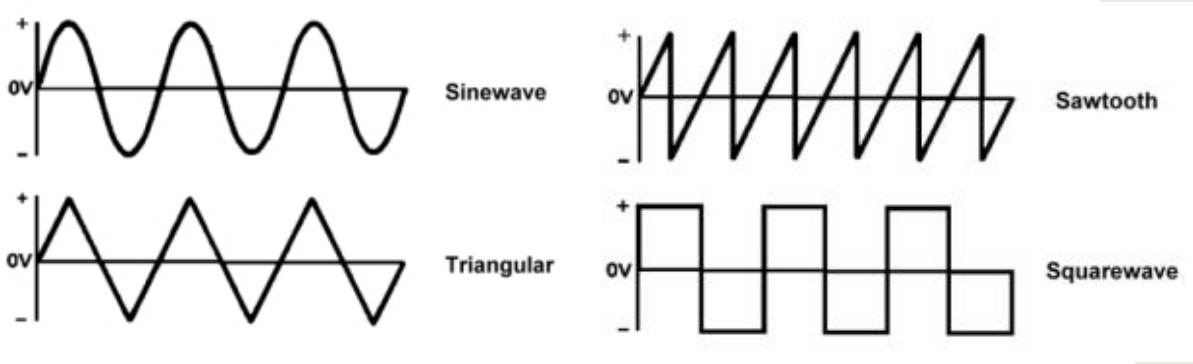
- Конструктори, що приймають лише необхідні від користувача параметри.

Користувач позбавлений потреби дотримуватися структури заголовків та обраховувати інші необхідні для валідності формату параметри;

```
WaveFormat::WaveFormat(int numberOfChannels, DWORD samplesRate, WORD bitsPerSample, DWORD channelMask) {

if (channelMask == 0) {

            fmt.wFormatTag = WAVE_FORMAT_PCM;
            fmt.nChannels = numberOfChannels;
            fmt.nSamplesPerSec = samplesRate;
            fmt.wBitsPerSample = bitsPerSample;
            fmt.nBlockAlign = numberOfChannels * bitsPerSample / 8;
            fmt.nAvgBytesPerSec = samplesRate * fmt.nBlockAlign;
            fmt.cbSize = 0;
        }
        else {

            fmtExtensible.Format.wFormatTag = WAVE_FORMAT_EXTENSIBLE;
            fmtExtensible.Format.nChannels = numberOfChannels;
            fmtExtensible.Format.nSamplesPerSec = samplesRate;
            fmtExtensible.Format.wBitsPerSample = bitsPerSample;
            fmtExtensible.Format.nBlockAlign = numberOfChannels * bitsPerSample / 8;
            fmtExtensible.Format.nAvgBytesPerSec = samplesRate * fmtExtensible.Format.nBlockAlign;
            fmtExtensible.Format.cbSize = sizeof(WAVEFORMATEXTENSIBLE) - sizeof(WAVEFORMATEX);
            fmtExtensible.Samples.wValidBitsPerSample = bitsPerSample;
            fmtExtensible.dwChannelMask = channelMask;
            fmtExtensible.SubFormat = KSDATAFORMAT_SUBTYPE_PCM;
        }
    }
```

- метод, що будує валідну RIFF структуру та записує її в файл, використовуючи заголовки форматів як fmt дані та згенерований сигнал, як підсекцію data. Таким чином користувач позбавлений необхідності власноруч будувати RIFF структуру для створення валідного файлу та побайтово записувати значення до файлу;

- методи, що генерують семпли базових звукових хвиль (синус, пилкоподібна, квадратна, трикутна), міксуванням яких в світі цифрових сигналів створюються більш складні мелодії;



- метод для зчитування заголовку wave файлу. Для користувача відсутня потреба зчитувати кожний елемент RIFF структури. Автоматично визначається, якого формату даний файл, і параметри записуються до відповідного екземпляру формату

- метод для зчитування даних семплів з файлу та запису в масив. Це дає можливість для подальшої обробки сигналу

```
template <typename SampleType>
SampleType* readWaveData(ifstream& input) {
    char subch2[4];
    DWORD subchunk2Size;

    input.read((char*)&subch2, 4); //data
    input.read((char*)&subchunk2Size, 4); //dataSize
    if (strncmp(subch2, "data", 4) != 0) {
        cerr << "Invalid data chunk" << endl;
    }
    dataSize = subchunk2Size / sizeof(SampleType);
    SampleType* samples = new SampleType[subchunk2Size / sizeof(SampleType)];

    input.read((char*)samples, subchunk2Size);
    return samples;
};
```

- метод для розділення масиву оцифрованих даних за каналами, що дає можливість обробляти потоки кожного каналу окремо;

```
void setNumberOfChannels(int numberOfChannels) {
    fmt.nChannels = numberOfChannels;
    fmt.nBlockAlign = numberOfChannels * fmt.wBitsPerSample / 8;
    fmt.nAvgBytesPerSec = fmt.nSamplesPerSec * fmt.nBlockAlign;
}

void setSamplesRate(DWORD samplesRate) {
    fmt.nSamplesPerSec = samplesRate;
    fmt.nAvgBytesPerSec = samplesRate * fmt.nBlockAlign;
}

void setBitsPerSample(WORD  bitsperSample) {
    fmt.wBitsPerSample = bitsperSample;
    fmt.nBlockAlign = fmt.nChannels * bitsperSample / 8;
    fmtExtensible.Samples.wValidBitsPerSample = bitsperSample;
}
```

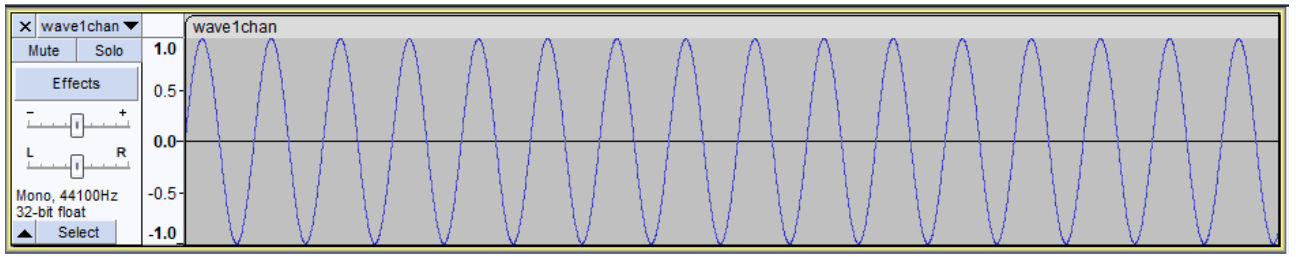
- getters, setters для доступу до параметрів заголовку.

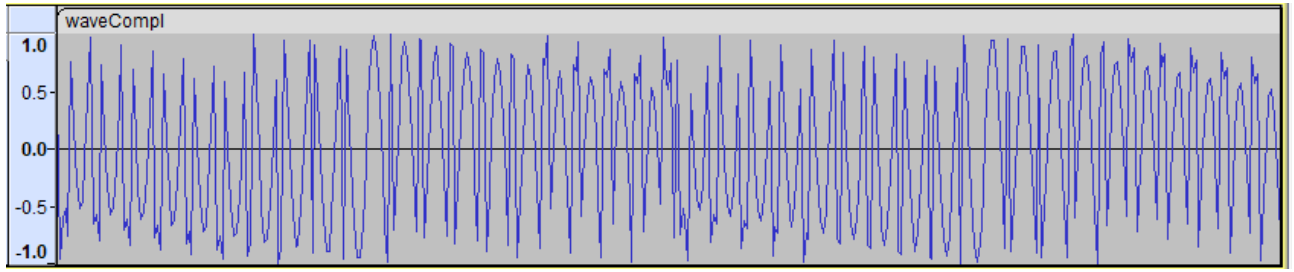
- метод для злиття масивів, що несуть інформацію для окремих каналів, в 1 масив, що може бути відтворений;

ПРИКЛАДИ ВИКОРИСТАННЯ

```
WaveFormat waveFormat(1, 44100, 16);        //fmt header
const int duration = 5;
double frequency = 200.0;

const int sampleRate = 44100;
const int numSamples = duration * sampleRate;
int16_t* data = new int16_t[numSamples];

for (int i = 0; i < numSamples; i++) {            //data generation
    double time = (double)(i) / sampleRate;
    data[i] = waveFormat.sinWave<int16_t>(frequency, time);
}
waveFormat.writeWaveFile("wave1chan.wav", numSamples * sizeof(int16_t), data, true); //writing to file
```

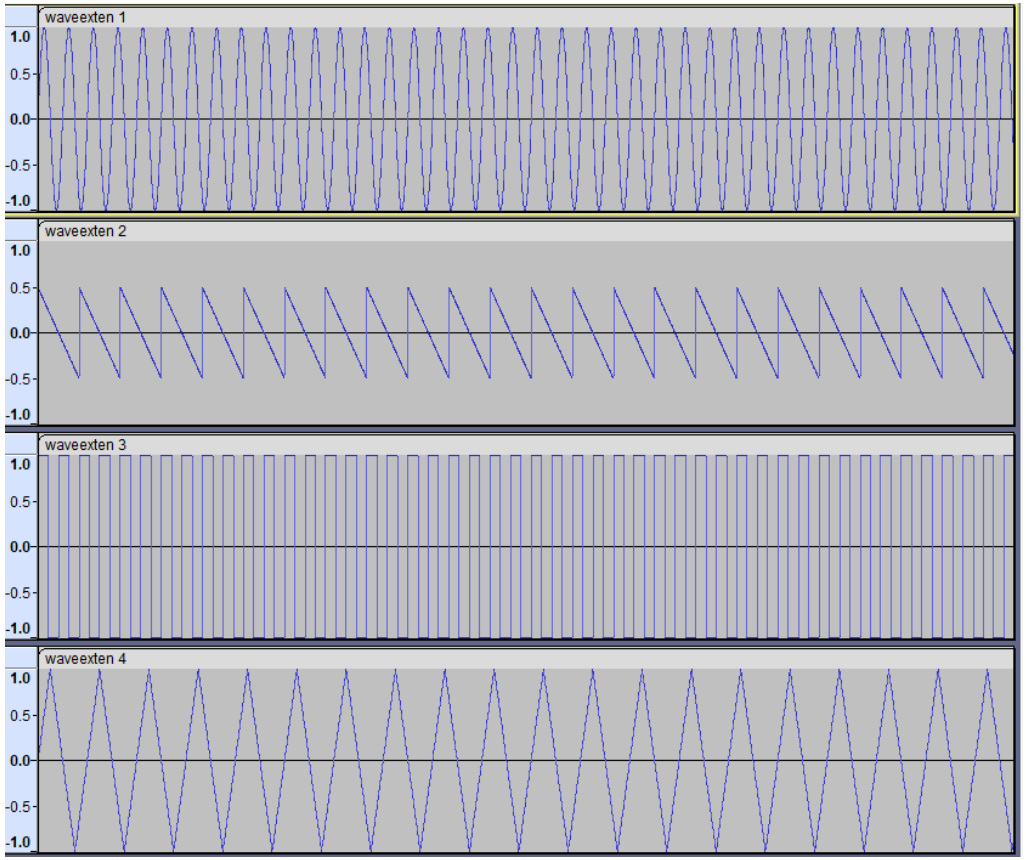
WAVE аудіо
1 канал,
44,100 частота дискретизації
16 біт



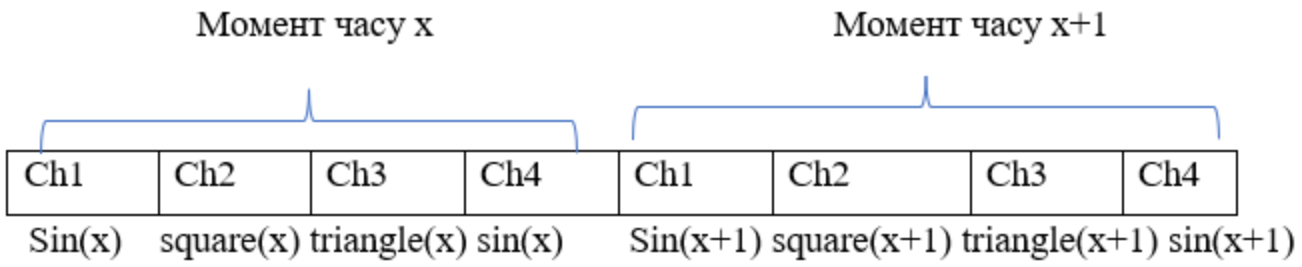



```
for (int i = 0; i < numSamples; i++) {

    double time = (double)(i) / sampleRate;

    data [i] = waveFormat.sawWave<int16_t>(frequency, time)+
waveFormat.sinWave<int16_t>(4000, time) ;

};
```

WAVEFORMATEXTENSIBLE, 4 КАНАЛИ, 48000 Гц, 24 біт




```
int nchannels = 4;

   DWORD samplesRate = 48000;

   int bitsPerSample = 24;

   DWORD channelMask = SPEAKER_FRONT_LEFT | SPEAKER_FRONT_RIGHT |
SPEAKER_FRONT_CENTER | SPEAKER_BACK_CENTER;

   WaveFormat waveFormat(nchannels, samplesRate, bitsPerSample, channelMask);
```



Де Ch1- SPEAKER_FRONT_LEFT, Ch2 - SPEAKER_FRONT_RIGHT,

Ch3 - SPEAKER_FRONT_CENTER, Ch4 - SPEAKER_BACK_CENTER, як вказано в maskChannel

ЗЧИТУВАННЯ ТА ОБРОБКА

Файл WAVE

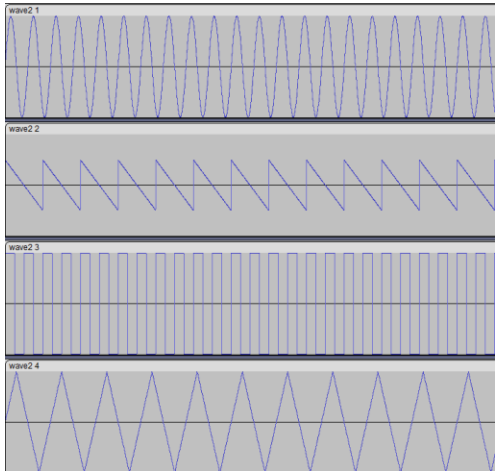
```
Format: 1
Number of Channels: 1
Sample Rate: 44100
Bit per Second: 16
Number of Samples: 220500
```

Файл WAVEFORMATEXTENSIBLE

```
Format: 65534
Number of Channels: 4
Sample Rate: 48000
Bit per Second: 32
Number of Samples: 240000
```

```
WaveFormat readFormatP;
ifstream fileReadP("waveexten16.wav", ios::binary);
readFormatP.readWaveHeader(fileReadP);
int16_t* readData = readFormatP.readWaveData<int16_t>(fileReadP);
fileReadP.close();
int arraySize = readFormatP.dataSize;
int channelsR = readFormatP.getChannels();

int16_t** chanArrays = new int16_t * [channelsR];
int* subarraySizes = new int[channelsR];
readFormatP.divideByChannels(channelsR, readData, arraySize, chanArrays, subarraySizes);
//getting separate arrays for each channel

int firstSample = readFormatP.getSamplesPerSec() * 1;
int lastSample = readFormatP.getSamplesPerSec() * 4;
int newSize = (lastSample - firstSample) * 4;          //getting number of samples for cutting

int16_t** cutArrays = new int16_t * [channelsR];
for (int i = 0; i < channelsR; ++i) {                  //creating new arrays, containing cutted versions
    cutArrays[i] = new int16_t[lastSample - firstSample];
    copy(chanArrays[i] + firstSample, chanArrays[i] + lastSample + 1, cutArrays[i]);
}

int16_t* samplesP = new int16_t[newSize];
readFormatP.mergeChannels(channelsR, cutArrays, (newSize / channelsR), samplesP, newSize);
//merging into 1 array, which can be written into file

readFormatP.writeWaveFile("wave2.wav", (newSize * sizeof(int16_t)), samplesP, false);
```

Довжина вхідного аудіо була 5 сек, вихідного – 3 секунди. Audacity дозволяє впевнитися, що звукові хвилі під час обробки не постраждали, графік вийшов ідентичним до вхідного.



ВИСНОВОК

В даній роботі були досліджені аспекти програмного створення та обробки аудіо формату WAVE та його розширення WAVEFORMATEXTENSIBLE. Було визначено особливості зберігання даних в файлі (RIFF структура), визначені якісні параметри кожної зі структур та їх можливі значення, приділено увагу генерації базових звукових хвиль.

Був проведений короткий огляд напрямків обробки аудіо та визначено особливості роботи з аудіо форматами звичайного та розширеного WAVE форматів.

В результаті роботи розроблений власний клас, який забезпечує зручний інтерфейс для роботи з обома структурами. Він оптимізований під роботу з аудіо з різними розрядностями, кількістю каналів, частотою дискретизації. Клас дозволяє користувачу виконувати створення, записування, зчитування, маніпулювання властивостями, розділення каналів для подальшої обробки аудіо файлів формату WAVE з мінімальними зусиллями