Signal Processing

CPE 381 Foundations of Signals & Systems for Computer Engineers

Dr. Emil Jovanov

Project: File I/O

- Provide file name
- command line arg

```c
int main(int argc, char* argv[])
    {
    /********************OPEN INPUT AND OUTPUT FILE****************************************/

    char inputFileName[255];
    char outFileName[255];
    static FILE* rFile;

    if(argc == 1){
        //Program will ask for the filename if filename is not specified
        printf("Please enter filename of file to be opened: ", argv[1]);
        scanf("%s",inputFileName);
        }
    else if(argc > 2)
        {
        //Program will not execute if there are too many parameters
        printf("Usage: %s [FILENAME]\n", argv[0]);
        exit(1);
        }

    //Open filename for binary input
    if (argc==1){
        rFile = fopen(inputFileName, "rb");
        }
    else{
        rFile = fopen(argv[1], "rb");
        }

    //If input file is not opened correctly will close the program
    if (rFile==NULL) {
        printf ("File error");
        system("pause");
        exit (1);
        }
```

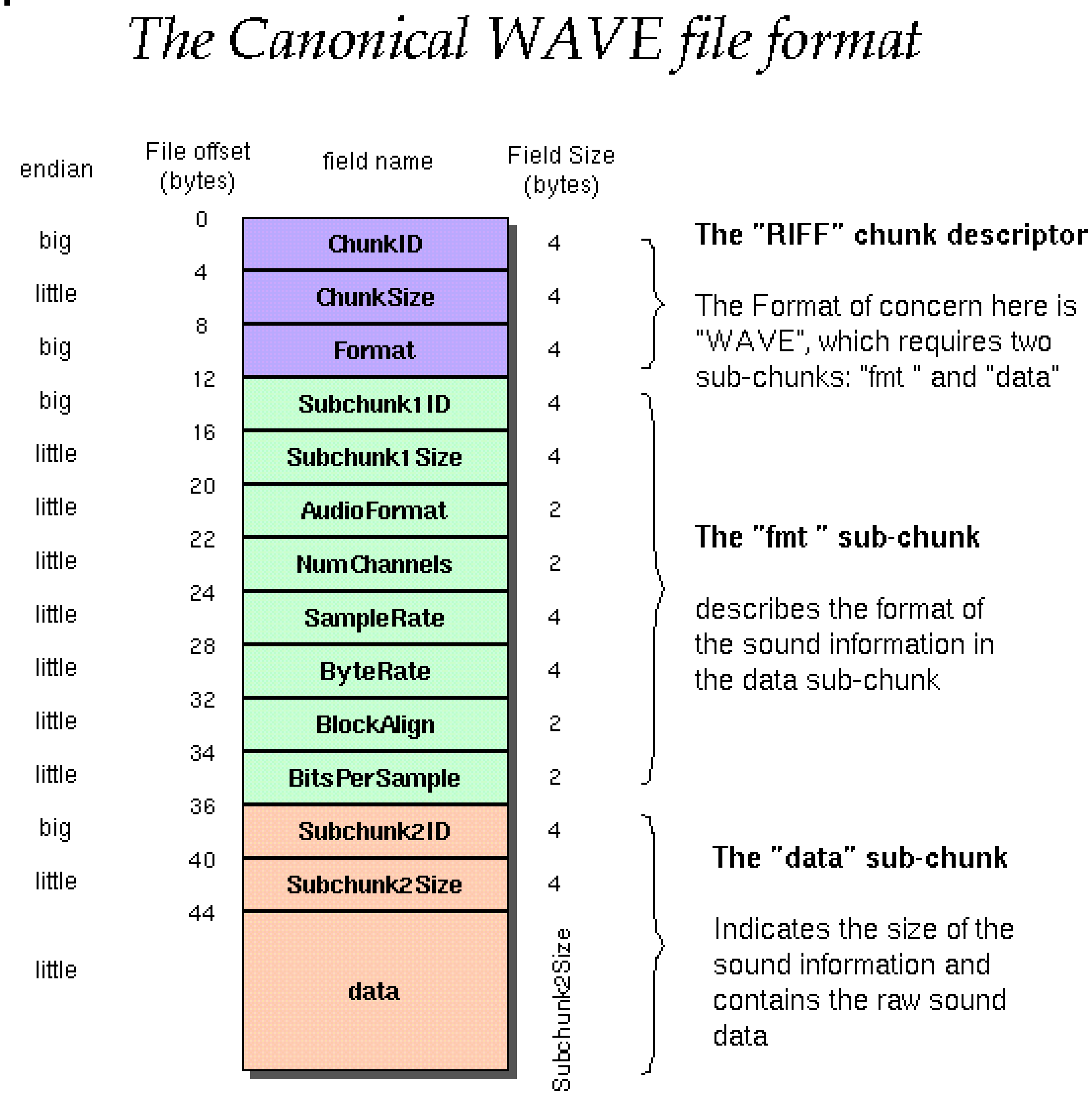
Project: File I/O

- get output file ready

```
//Printing the name of the input file to the console
printf("Input file: %s\n", argv[1]);

//Creating filename for output file
strncpy (outFileName,argv[1], strlen(argv[1])-4);
strcpy (outFileName+(strlen(argv[1])-4),"_downsample.wav");
}
```

Reading formatted files #1

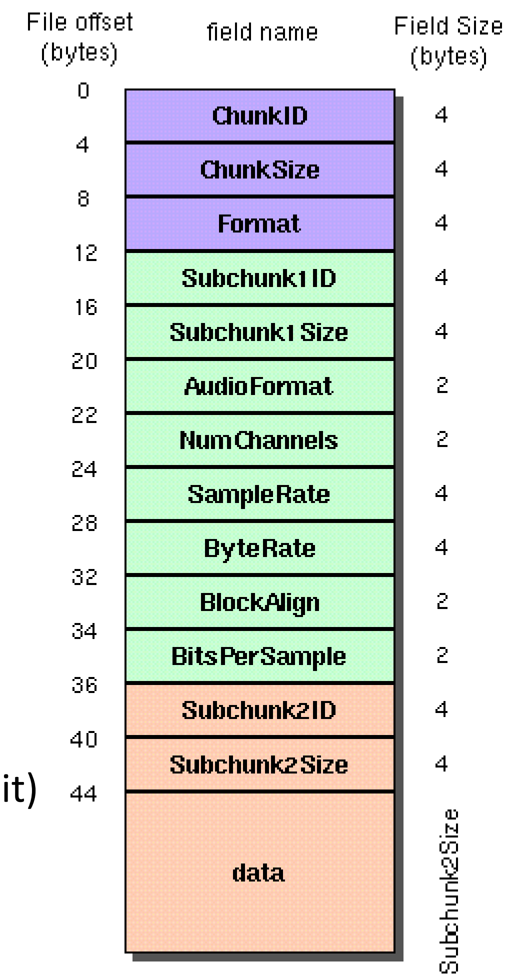
- WAV file header



Reading formatted files #1

- reading file header

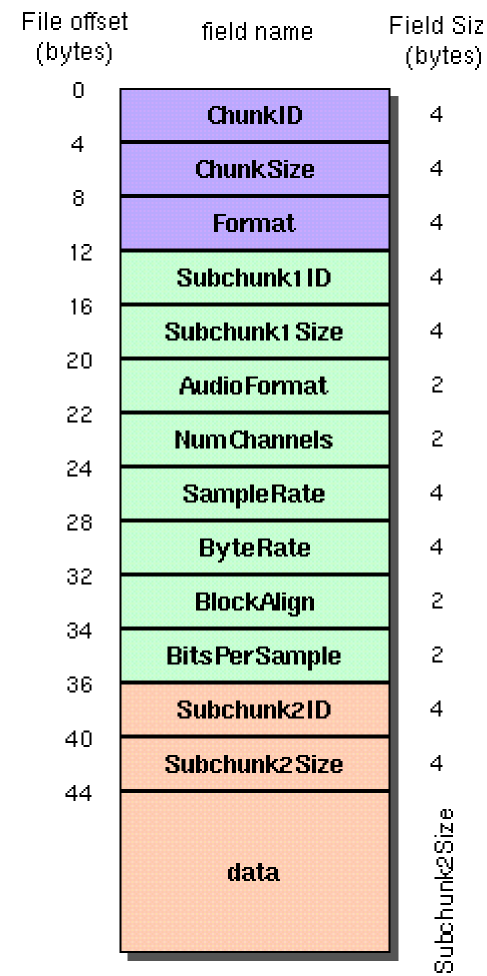
```
//WAV Header File Info
const int HEADER_SIZE = 44;
const int SAMP_RATE_OFFSET = 24;
const int BPS_OFFSET = 34;
const int NUM_CH_OFFSET = 22;
const int AUD_FORM_OFFSET = 20;
const int SUB_CHANK_SIZE_OFFSET = 40;
const int BYTE_RATE_OFFSET = 28;

//The header for PCM is always 44 bytes long
fread(rBuffer, sizeof(char), HEADER_SIZE*sizeof(char), rFile);

sampleSize = *(short*)(rBuffer+BPS_OFFSET);//size of each sample (8/16 bit)
subChunkSize = *(unsigned long*)(rBuffer+SUB_CHANK_SIZE_OFFSET);

//Holds size in bytes the samples take up in the file
sampleRate  = *(long*)(rBuffer+SAMP_RATE_OFFSET);
audioFormat = *(short*)(rBuffer+AUD_FORM_OFFSET);
numOfChannels = *(short*)(rBuffer+NUM_CH_OFFSET);
byteRate = *(long*)(rBuffer+BYTE_RATE_OFFSET);
```



Reading formatted files #2

- define structures

```
struct WavHeader
        {
        unsigned long  ChunkID ;     // the letters "RIFF" in ASCII form
        unsigned long ChunkSize;     // This is the size of the entire file in bytes
                // minus 8 bytes for the two fields not included in this count: ChunkID and ChunkSize.
        unsigned long Format;                //Contains the letters "WAVE"
        unsigned long Subchunk1ID;           //Contains the letters "fmt "
        unsigned long Subchunk1Size;         //16 for PCM
        unsigned short AudioFormat;          //PCM = 1 (i.e. Linear quantization)
                // Values other than 1 indicate some form of compression.
        unsigned short NumChannels;          //Mono = 1, Stereo = 2, etc.
        unsigned long SampleRate;            //8000, 44100, etc.
        unsigned long ByteRate;      //SampleRate * NumChannels * BitsPerSample/8
        unsigned short BlockAlign;   //NumChannels * BitsPerSample/8
        unsigned short BitsPerSample;        //    8 bits = 8, 16 bits = 16, etc.
        unsigned long Subchunk2ID;     //  Contains the letters "data"
        unsigned long Subchunk2Size;  //NumSamples * NumChannels * BitsPerSample/8
        };
```



Reading formatted files #3

- … and use structures

```
struct WavHeader
{
        // ...
};

struct WavHeader fileHeader;    //structure fot header of WAV file

fread(&fileHeader, sizeof(WavHeader),1, rFile); //Read header into structure

//Number of samples in the file (total number, sum of number of samples from each channel)
sampleCount = fileHeader.Subchunk2Size /
        ((fileHeader.BitsPerSample /8) * fileHeader.NumChannels);

fileHeader.SampleRate = fileHeader.SampleRate>>1;
fileHeader.Subchunk2Size = fileHeader.Subchunk2Size>>1;
fileHeader.ByteRate = fileHeader.ByteRate>>1;

//Write header to modified output file
fwrite(&fileHeader, sizeof(WavHeader), 1, wFile);
```

Processing

- … sample by sample
  - example: mono/16 bit

```
fread(&inBufferCur, sizeof(short), 1, rFile); //Get the first sample

while(!feof(rFile))
	//Take the average of the current and previous sample, and write it to the output file
	{
	if(count%2)
		{
		temp = inBufferPrev+inBufferCur;
		outBuffer =(short)(temp>>1);
		//Write the result to the output file, one at a time per iteration
		fwrite(&outBuffer, sizeof(short), 1, wFile);
		}

	inBufferPrev = inBufferCur; //Copy current sample into previous

	fread(&inBufferCur, sizeof(short), 1, rFile); //Get next sample

	count++;
	}
}
```

Performance measurement

- profile critical sections of the code

```
//*** PERFORMANCE MEASUREMENT *************//

//Get the starting time
clock_t time_start = clock();

        // do something

printf("Processing time: %6.2fs\n", (double)(clock() - time_start)/CLOCKS_PER_SEC);
```

Filtering

- Init


```c
// input & output samples
#define FILT_LEN 12

int NB=FILT_LEN;                    // filter length


/*** Filter initialization ***/
void filt_init_var(int *x, int *y) {
  register int ii;

  for (ii=0; ii<FILT_LEN; ii++)
    x[ii] = y[ii] = 0;
}
```

Filtering

- FIR filter (floating point)

```
void xiir_filter(int * x, int * y, int sample)
{
        /* fixed point filter procedure
                xin - input signal
                yout - filtered input signal
        */
        long templ;
        register int i;

        /* the latest sample is at index 0, all other are shifted */
        for (i=NB-1;i>0;i--) {
                x[i]=x[i-1];
                y[i]=y[i-1];
        }
        x[0]=sample;

// FIR filter
        templ=0;
        for (i=0;i<NB;i++) {
                templ += x[i]*B[i];
                }

        y[0]=(int)templ;
}
```

Fixed Point Filtering

- IIR filter

```
void xiir_filter(int * x, int * y, int sample) {
        /* fixed point filter procedure
                xin - input signal
                yout - filtered input signal
        */
        long templ;
        register int ii;

        /* the latest sample is at index 0, all other are shifted */
        for (ii=NB-1;ii>0;ii--) {
                x[ii]=x[ii-1];
                y[ii]=y[ii-1];
        }
        x[0]=sample;

        /*** B coefficients */
        templ=0;
        templ += (38740 * x[0]) >> 6;           /* b(1) -> 0.009236 */
        templ += (38740 * x[1]) >> 5;           /* b(2) -> 0.018472 */
        templ += (38740 * x[2]) >> 6;           /* b(3) -> 0.009236 */

        /*** A coefficients */
        templ += (56044 * y[1]) << 1;           /* a(2) -> -1.710329 */
        templ -= (48973 * y[2]);        /* a(3) -> 0.747274 */

        y[0]=templ >> 16;
}
```