استاد: دکتر مهدی کارگهی طراحان: محمد امانلو، سینا طبسی

هدف تمرین

هدف از این تمرین آشنایی شما با مفاهیم اولیه طراحی چند ریسه‌ای[1] یک مسئله است.
در این تمرین شما به اعمال فیلترهایی روی یک قطعه صوتی می‌پردازید. این قطعه صوتی در فرمت Wav به همراه کد نحوه خواندن این فایل به شما داده می‌شود و شما باید با اعمال فیلترها در هر دو حالت موازی و سری، سرعت اجرای هر یک از این دو مورد را باهم بررسی کنید.

شرح تمرین

در این پروژه، هدف طراحی و پیاده‌سازی یک سیستم پردازش سیگنال صوتی است که قادر است فیلترهای مختلف دیجیتال را روی سیگنال‌های صوتی اعمال کند. فیلترهای مد نظر شامل 4 فیلتر **Band-pass filter، Notch filter, FIR filter** و **IIR filter** هستند که به ترتیب برای عبور فرکانس‌های درون یک بازه خاص، حذف فرکانس‌های خاص، فیلتر پاسخ ضربه‌ای محدود و فیلتر پاسخ ضربه‌ای بی‌نهایت طراحی شده‌اند.

در ابتدای کار، فایل صوتی ورودی از طریق رابط کاربری بارگذاری می‌شود. سپس، سیستم سیگنال صوتی را تجزیه و تحلیل کرده و فرکانس‌های مختلف سیگنال را استخراج می‌کند. در ادامه، بسته به نوع فیلتر انتخاب شده، سیگنال‌های موردنظر برای عبور یا حذف از آن پردازش می‌شوند و نتیجه به سیگنال جدید تبدیل می‌شود. در نهایت، سیگنال فیلتر شده به صورت فایل جدید ذخیره می‌شود.

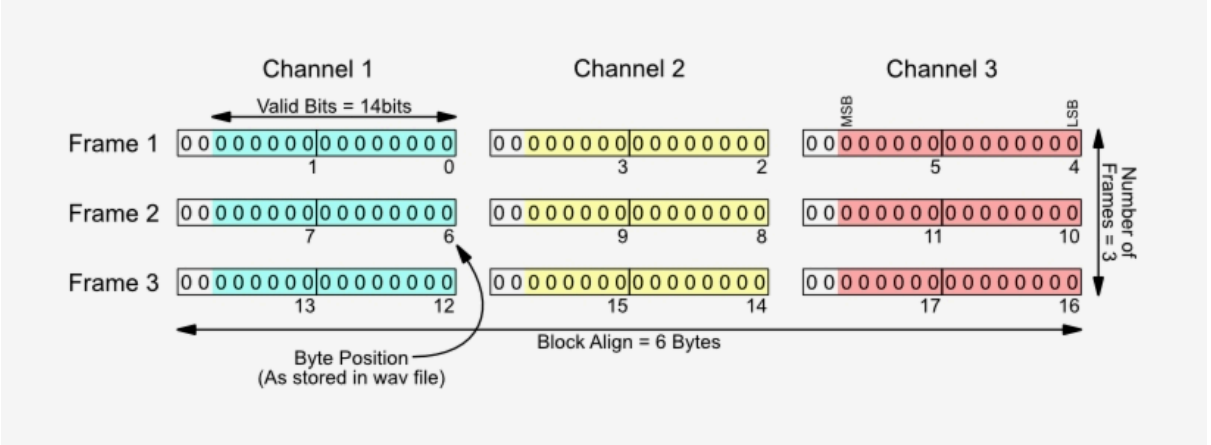
خواندن صوت

کد این قسمت در کنار فایل صورت پروژه به شما داده می‌شود. امکان ایجاد هرگونه تغییر در فایل کمکی پروژه وجود دارد. این فایل صرفا نقش کمک‌کننده برای انجام پروژه را دارا است. برای خواندن و ذخیره فایل‌های صوتی در این پروژه از کتابخانه **libsndfile** استفاده شده است. برای آشنایی بیشتر با این کتابخانه می‌توانید از این لینک استفاده کنید. این کتابخانه به‌طور گسترده برای پردازش فایل‌های صوتی در فرمت‌های مختلف مانند **WAV** مورد استفاده قرار می‌گیرد. برای آشنایی بیشتر با

[1] Multi threading

نحوه ذخیره سازی در این فرمت از این لینک می‌توانید استفاده کنید. در این پروژه، ابتدا کاربر می‌تواند یک فایل صوتی را از طریق رابط کاربری یا خط فرمان بارگذاری کند. فایل‌های صوتی وارد شده به برنامه از طریق کتابخانه **libsndfile** بارگذاری می‌شوند. در ابتدا، سیگنال صوتی از فایل ورودی خوانده شده و به داده‌های دیجیتال (آرایه‌ای از نمونه‌ها) تبدیل می‌شود. این داده‌ها سپس به‌صورت متغیرهای داخلی برنامه ذخیره می‌شوند تا مراحل پردازش و فیلتر کردن روی آن‌ها انجام گیرد. فرمت این ذخیره‌سازی بر عهده خود شماست.

همچنین یک فایل صوتی بعنوان نمونه در کنار پروژه برای شما قرار داده شده‌است. لزومی به استفاده از همین فایل ندارید می‌توانید هر فایل صوتی دیگر با فرمت wav را برای انجام پروژه انتخاب کنید.



فیلترها

1. **فیلتر باند گذر(Band-pass Filter)**

فیلتر باندگذر یک نوع فیلتر است که به شما این امکان را می‌دهد که فقط فرکانس‌های خاصی را از یک سیگنال عبور دهید و بقیه فرکانس‌ها را حذف کنید. این فیلتر به‌ویژه زمانی مفید است که بخواهید یک بخش خاص از فرکانس‌ها را از سیگنال جدا کنید و فرکانس‌های دیگر را نادیده بگیرید. برای مثال، در پردازش صوتی، می‌توان از فیلتر باندگذر برای استخراج صدای خاص از میان نویزها یا سیگنال‌های دیگر استفاده کرد. فیلتر باندگذر به‌طور خاص برای عبور دادن فرکانس‌ها در یک بازه مشخص از فرکانس‌ها طراحی می‌شود که این فیلتر دو محدوده دارد:

1. فرکانس پایین‌تر از یک مقدار خاص که فیلتر می‌شود.
2. فرکانس بالاتر از یک مقدار خاص که فیلتر می‌شود.

بنابراین، این فیلتر فقط فرکانس‌هایی را که در بین دو حد مشخص قرار دارند، عبور می‌دهد و سایر فرکانس‌ها را مسدود می‌کند. برای کسب اطلاعات بیشتر در خصوص این نوع فیلتر، می‌توانید از این لینک و در خصوص توضیحات نموداری این فیلتر نیز می‌توانید از این لینک استفاده نمایید.

فیلتر باندگذر را می‌توان به‌صورت ریاضی به شکل روبرو نوشت: $$H(f) = \frac{f^2}{f^2 + (\Delta f)^2}$$

در این فرمول:

- $H(f)$ نشان‌دهنده پاسخ فیلتر به فرکانس f است.
- f فرکانس سیگنال است (یعنی چه میزان از فرکانس‌ها به فیلتر داده شده است).
- Δf عرض باند یا پهنای باند است که نشان می‌دهد این فیلتر چه محدوده‌ای از فرکانس‌ها را عبور می‌دهد.

2. فیلتر نوچ(Notch Filter)

فیلتر نوچ به‌گونه‌ای طراحی شده است که فقط یک فرکانس خاص را مسدود کند، در حالی که سایر فرکانس‌ها را دست‌نخورده باقی می‌گذارد. این فرکانس خاص معمولاً نویز یا اختلالات الکتریکی هستند که در سیگنال وجود دارند. برای مثال، در محیط‌های صنعتی یا هنگام ضبط صدا، ممکن است فرکانس‌هایی مانند 50 هرتز (که ناشی از نویزهای الکتریکی در سیستم برق است) وجود داشته باشد. فیلتر نوچ این فرکانس خاص را حذف می‌کند و باعث می‌شود که دیگر اختلالات موجود در سیگنال باقی بمانند. برای اطلاعات بیشتر در خصوص این نوع فیلتر، می‌توانید از این لینک استفاده نمایید.

فیلتر نوچ را می‌توان به‌صورت ریاضی با فرمول روبرو مدل‌سازی کرد: $$H(f) = \frac{1}{(\frac{f}{f_0})^{2n}+1}$$

در این فرمول:

- $H(f)$ نشان‌دهنده پاسخ فیلتر به فرکانس f است.
- f فرکانس سیگنال است (یعنی چه میزان از فرکانس‌ها به فیلتر داده شده است).
- f_0 فرکانس خاصی است که می‌خواهیم حذف کنیم (این همان فرکانس نویز یا اختلال است).

3. فیلتر (Finite Impulse Response)

فیلتر FIR یا فیلتر پاسخ ضربه‌ای محدود، یکی از ساده‌ترین و پرکاربردترین فیلترهای دیجیتال است. ویژگی اصلی این فیلتر این است که در آن پاسخ سیستم به یک ضربه (Impulse Response) در زمان گسسته، محدود است و در نتیجه به آن فیلتر "پاسخ ضربه‌ای محدود"

گفته می‌شود. این فیلتر برخلاف فیلترهای IIR که از بازخورد استفاده می‌کنند، هیچ حلقه بازخوردی ندارد و بنابراین هیچ‌گونه تغییری از سیگنال‌های قبلی در آن لحاظ نمی‌شود.

پاسخ ضربه‌ای محدود یعنی اینکه این فیلتر تنها به ورودی‌های گذشته (و نه به ورودی‌های آینده) وابسته است و هر ورودی جدید تنها تأثیری محدود بر خروجی دارد. برای اطلاعات بیشتر در خصوص این فیلتر، می توانید از این لینک استفاده کنید.

فرمول ریاضی فیلتر FIR به این شکل است که سیگنال خروجی به‌عنوان مجموع ضرایب فیلتر و ورودی‌های گذشته به‌دست می‌آید. به عبارت دیگر، خروجی فیلتر در هر لحظه برابر است با مجموع ورودی‌های گذشته که ضرب در وزن‌های فیلتر (یا همان پاسخ ضربه‌ای) شده‌اند:

$$[k - x[n \cdot h[k] \sum_{k=0}^{M-1} = y[n]$$

در این فرمول:

- $y[n]$: خروجی فیلتر در زمان n
- $k - n$ $k - x[n]$: ورودی فیلتر در زمان
- $h[k]$: ضرایب فیلتر که به‌طور معمول به‌عنوان پاسخ ضربه‌ای فیلتر شناخته می‌شوند.
- M: طول فیلتر که تعداد ضرایب فیلتر را مشخص می‌کند.

در اینجا، برای هر مقدار از زمان n، خروجی فیلتر به مجموع حاصل‌ضرب ورودی‌های قبلی در ضرایب مربوطه‌شان بستگی دارد.

- در فیلتر FIR، سیگنال خروجی $y[n]$ با توجه به ورودی‌های قبلی و ضرایب فیلتر محاسبه می‌شود.
- برای پیاده‌سازی یک فیلتر FIR، باید ضرایب $h[k]$ که به‌طور معمول به عنوان پاسخ ضربه‌ای فیلتر شناخته می‌شود، مشخص شوند.
- این ضرایب معمولاً به‌صورت دستی به صورت موثر در نظر بگیرید.

4. فیلتر Infinite Impulse Response

فیلتر IIR یا فیلتر پاسخ ضربه‌ای بی‌نهایت یک نوع فیلتر دیجیتال است که برخلاف فیلترهای FIR از بازخورد (feedback) استفاده می‌کند. این فیلترها برای محاسبه سیگنال خروجی نه تنها از ورودی‌های فعلی و گذشته بلکه از خروجی‌های قبلی هم استفاده می‌کنند. به عبارت دیگر، در

فیلترهای IIR، خروجی‌ها به خودشان وابسته هستند و این وابستگی می‌تواند به‌طور بی‌نهایت ادامه یابد. این ویژگی باعث می‌شود که فیلترهای IIR برخلاف فیلترهای FIR که در آن‌ها تنها ورودی‌های گذشته استفاده می‌شود، پیچیده‌تر عمل کنند.

در فیلترهای IIR، سیگنال خروجی نه تنها از ورودی‌های فعلی و قبلی، بلکه از خروجی‌های قبلی نیز تاثیر می‌پذیرد. این باعث می‌شود که پاسخ فیلتر به یک ضربه (impulse) به‌طور بی‌نهایت در زمان ادامه پیدا کند. به عبارت دیگر، برای محاسبه سیگنال خروجی، علاوه بر ضرب ورودی‌ها در ضرایب مربوطه، خروجی‌های قبلی هم در محاسبات دخیل هستند. در خصوص اطلاعات بیشتر در مورد این فیلتر، می‌توانید از این لینک استفاده کنید.

فرمول کلی برای فیلتر IIR به‌صورت زیر است:

$$[j - y[n \cdot a[j] \sum_{j=1}^{N-1} - [k - x[n \cdot b[k] \sum_{k=0}^{M-1} = y[n]$$

در این فرمول:

- $y[n]$: خروجی فیلتر در زمان n
- $x[n - k]$: ورودی فیلتر در زمان $n - k$
- $b[k]$: ضرایب باند عبوری (feedforward) که بر ورودی‌ها اعمال می‌شوند.
- $a[j]$: ضرایب بازخوردی (feedback) که بر خروجی‌های قبلی اعمال می‌شوند.
- M: طول بخش ورودی (تعداد ضرایب feedforward).
- N: طول بخش بازخورد (تعداد ضرایب feedback).
- ورودی‌ها و خروجی‌های قبلی: در این فیلتر، سیگنال خروجی به‌طور مستقیم به ورودی‌های قبلی و همچنین خروجی‌های قبلی وابسته است. به این ترتیب، اثرات ورودی‌های گذشته تا مدت‌ها در خروجی مشاهده می‌شود.
- بازخورد: بازخورد در این فیلترها به این معنی است که سیگنال خروجی به ورودی‌های بعدی تاثیر می‌گذارد. این ویژگی می‌تواند به فیلتر کمک کند تا رفتار پیچیده‌تری ایجاد کند و ویژگی‌هایی مانند سریع‌تر پاسخ دادن یا پاسخ‌های طولانی‌تر ایجاد کند.

پیاده‌سازی سری

در این بخش از تمرین شما به پیاده‌سازی serial برنامه خواسته شده می‌پردازید. سعی کنید در این بخش از تمرین، بهترین پیاده‌سازی را از لحاظ زمان اجرا انجام دهید؛ چرا که برای مقایسه عملکرد پیاده‌سازی چندریسه‌ای با سری، حالت سری باید در حالت بهینه پیاده‌سازی شده باشد. پس از این مرحله اعمالی که بیشترین زمان اجرا را به خود اختصاص دادهاند را شناسایی کنید

پیاده‌سازی چند ریسه‌ای

در این بخش از تمرین به موازی سازی اعمال صورت گرفته در توابعی که در بخش قبل به عنوان Hotspot (توابعی که بیشترین زمان اجراها را به خود اختصاص میدهند) از آنها یاد کردید میپردازید. خواندن ورودی و ذخیره سازی آن در حافظه از اعمال زمان گیر در بسیاری از برنامه هاست که احتمالا از توابع مربوط به آنها (در کنار سایر توابع) به عنوان Hotspot های برنامه یاد کرده اید برای موازی سازی این بخش میتوانید خواندن و ذخیره سازی مقادیر فریم های صوت و اعمال فیلتر روی آنها را توسط چندین ریسه انجام دهید. (بهبود در زمان خواندن یا نوشتن فایل صوتی به وسیله چند ریسه‌ای کردن شامل نمره امتیازی خواهد بود.) بهترین ترکیب تعداد ریسه‌ها نحوه تقسیم داده ها و مکانیسم های همگام سازی ریسه ها را باید بدست آورده و انتخابهای خود را توجیه کنید در انتها میزان تسریع پیاده سازی چندریسه‌ای به پیاده سازی سری را از رابطه زیر بدست آورده و طبق قالب خروجی که در ادامه آمده است، گزارش کنید:

$$speedup = \frac{serial\ execution\ time}{parallel\ execution\ time}$$

دقت کنید که خروجی برنامه چندریسه‌ای شما باید مطابق با خروجی برنامه سری شما باشد. توجه شود که این بخش از تمرین باید به صورت چندریسه‌ای پیاده سازی گردد و سایر پیاده سازی ها قابل قبول نیست. دقت شود برای موازی سازی پروژه تنها مجاز به استفاده از کتابخانه pthreads هستید و استفاده از کتابخانه های دیگر به جز کتابخانه های پایه زبان C++ مجاز نیست.

ورودی و خروجی برنامه

برنامه شما باید نام فایل صوتی ورودی را از خط فرمان دریافت کند. نمونه اجرای برنامه با فرض اینکه صوت ورودی با نام input.wav در کنار فایل اجرایی شما قرار گرفته است در زیر آمده است.

```
./VoiceFilters.out input.wav
```

برای هر دو پیاده سازی سری و چندریسه‌ای باید صوت‌های خروجی به ازای هر فیلتر در فایلی به نام output[FilterName][Parallel/Serial].wav ذخیره شده و زمان اجرای خواندن صوت، زمان اجرای هر فیلتر و کل زمان طی شده در برنامه پس از اجرا چاپ شود. یک نمونه خروجی برنامه در زیر آمده است

```
Read: 4.751 ms
Band-pass Filter: 6.483 ms
Notch Filter: 0.127 ms
IRR Filter: 4.250
FIR Filter: 6.750
Execution: 89.168 ms
```

نکات و نحوهٔ تحویل

- . تمام خروجی‌های برنامه را در جریان خروجی استاندارد (stdout) چاپ کنید و فقط صوت نتیجه به عنوان فایلی جدا تولید میشود.
- برای انجام این پروژه ترجیحا از سیستم‌عامل‌های Unix-based استفاده کنید.
- تضمین میشود که ورودی‌هایی که به برنامه شما داده می‌شود صحیح هستند و نیازی به بررسی صحت ورودی توسط برنامه شما نیست. طراحی درست کارایی[2] برنامه و شکستن برنامه به بخشهای کوچکتر تأثیر زیادی در نمره تمرین دارد.
- دقت کنید که فایل آپلودی شما با نام OS_CA3_<SID>.zip حتما باید شامل دو پوشه مجزا باشد که در یک پوشه پیاده سازی سری (پوشه serial) و در پوشه دیگر پیاده سازی موازی (پوشه parallel) آورده شده است.
- دقت کنید که فایل zip شما شامل فولدر بیرونی نباشد و مستقیماً پس از unzip کردن آن، دو پوشه ذکر شده پیاده سازی سریال و موازی شما بدست آید.
- صوت ورودی و خروجی را در فایل آپلودی خود قرار ندهید. برای مثال یک نمونه فایل آپلودی مورد قبول در زیر آمده است:

[2] Performance

```
OS_CA3_81019xxxx.zip
├── parallel
|   ├── main.cpp
|   └── makefile
└── serial
    ├── main.cpp
    └── makefile
```

- برنامه شما باید در سیستم عامل لینوکس و با مترجم g++ با استاندارد C++11 ترجمه و در زمان معقول برای ورودی های آزمون اجرا شود.
- دقت کنید که پروژه شما باید دارای Makefile باشد. همچنین در Makefile خود مشخص کنید که از استاندارد C++ 11 استفاده میکنید.
- نام فایل اجرایی شما که در کنار Makefile خود ساخته میشود باید VoiceFilters.out باشد. نکته هایی که در جلسه توجیهی تمرین گفته میشود و یا در فروم های مربوطه مطرح می شوند بخشی از تمرین هستند؛ بنابراین به آنها توجه داشته باشید.
- هدف این تمرین یادگیری شماست. لطفاً تمرین را خودتان انجام دهید در صورت کشف تقلب مطابق قوانین درس با آن برخورد خواهد شد.
- در صورت داشتن سوال می‌توانید از طریق فروم درس یا شرکت در جلسات رفع اشکال سوالات خود را مطرح کنید.

موفق باشید!