C 언어를 이용한 컴퓨터 실습

Computer Exercises based on C Language

남승현

2019년 5월 3일

EE PCU

차 례

표 차례

그림 차례

서문

이 교재는 전자공학과 학생들이 C언어 기초를 수강한 다음 전자공학 분야에서 C언어를 활용하는 능력을 배양하는데 도움을 주기 위한 목적으로 만들어졌다. C 언어의 기본적인 내용을 직접 다루지만 않지만, 평소 자신이 공부한 C 언어 교재를 참고서로 놓고 참조하면서 이 교재를 공부하면 좋을 것이다.

사실 C 언어는 다른 사람들의 프로그램을 따라하면서 배우는 과목이다. 프로그램 작성 기술은 과목을 수강하고 외워서 배워지는 것이 아니라 자신이 직접 여러 프로그램들을 작성하고 다른 사람들의 프로그램을 보면서 저절로 습득될 수 있다. 중요한 것은 좋은 프로그램 습관을 익히는 것이다.

이 교재를 이용하면서 몇가지 중요한 약속을 하기로 한다.

- 먼저 프로그램을 직접 자신이 처음부터 작성하는 습관을 익혀야 한다. 다른 사람의 프로그램을 복사-편집하는 것으로는 실력이 늘지 않는다.
- 그러므로 이 과목을 수강하는 학생들은 자신이 직접 숙제나 과제들을 작성하는데 시간을 아끼지 말아야 한다. 프로그램 작성 기술은 시간이 아주 많이 소요되는 기술이다. 시간 투자를 할 생각이 없는 학생은 이 과목을 포기하는 것이 좋다.
- 좋은 프로그램을 작성하려면 프로젝트의 이름, 소스 파일의 이름, 변수의 이름을 의미있게 잘 결정해야 한다. 귀찮다고 아무런 의미없는 변소나 파일 이름을 사용하는 습관은 매우 잘못된 것이다. 이 교재에서 특별히 언급된 프로젝트 이름, 파일 이름, 변수 이름 등은 반드시 그 이름을 사용하도록 하라.
- 프로그램이 암기 과목이 아님에도 불구하고 몇가지 기본적인 내용들은 반드시 암기하는 것이 필요하다. 대부분의 프로그램

내용과 형식들은 자주 사용하면서 저절로 암기된지만, 처음에는 적극적으로 몇가지를 암기하는 것이 필요하다.

- 프로그램은 항상 오류를 수정하는 과정, 즉 디버그(debug)과정이 필요하다. 프로그램의 크기가 커짐에 따라 디버그 과정을 매우 중요하다. 그러므로 사용하는 통합개발도구(IDE)의 – 우리의 경우 마이크로소프트 비주얼 스튜디오 익스프레스 – 디버그 용법을 잘 알아 두면 편리하다.

1
좌표계와 좌표 변환

2차원 공간에서 사용되는 두 가지 좌표계는 직교좌표계와 극좌표계이다.

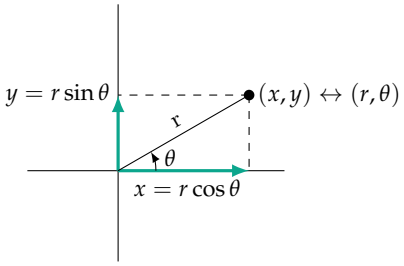


그림 1.1 직교 좌표와 극 좌표

1.1 각도의 표현 방법

C 프로그램에서 삼각함수를 사용할때에는 라디안(radian) 단위를 사용한다. 라디안은 일반적으로 0에서 2π 혹은 $-\pi$에서 π의 값으로 표현된다. 그러나 사람은 라디안 값을 직관적 공간적으로 사고하기 어렵기 때문에 각도로 0에서 360° 혹은 −180°에서 180°의 범위 값으로 사용하는 경우가 많다. 즉,

$$\pi/4 \longleftrightarrow 45^\circ \qquad \pi/3 \longleftrightarrow 60^\circ$$
$$\pi \longleftrightarrow 180^\circ \qquad -\pi \longleftrightarrow -180^\circ$$

각도와 라디안의 관계는 다음과 같다.

$$\theta_{rad} = \theta_{deg}\left(\frac{\pi}{180}\right) \qquad \theta_{deg} = \theta_{rad}\left(\frac{180}{\pi}\right)$$

각도를 입력받아 라디안으로 변환하여 출력하는 프로그램을 작성하는 과정은 다음과 같다.

1. 먼저 `deg2rad`라는 이름의 프로젝트를 새로 만든다. (절대 다른 이름을 사용하지 말라!) 이때 CompSim이라는 솔루션 안에 만들어야 한다.
2. 프로젝트 안에 `deg2rad.c`라는 이름의 새로운 소스 코드를 생성한다.
3. `theta_rad`, `theta_deg`를 정의한다. 여기서는 `double`형 변수로 정의한다.
4. `PI`는 π를 의미하는 값으로 상수로 정의하는 것이 좋다[1]. 다음 두[2] 방법 중 하나를 사용할 수 있다.
   - `#define PI 3.1415926535`
   - `const double PI = 3.1415926535;`
5. 프로그램이 `theta_deg`를 입력받아 `theta_rad`를 출력하도록 작성하라.

[1] C 언어에서 상수 이름을 명명할때는 일반적으로 모두 대문자를 사용한다.
[2] 한 가지 방법이 더 있는데 이는 나중에 소개될 것이다.

위의 과정을 거쳐 완성된 프로그램은 다음과 같다.

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <math.h>

const double PI = 3.1415926535;

int main(void)
{
    double theta_deg, theta_rad;

    printf("theta_deg ? ");
    scanf("%lf", &theta_deg);

    theta_rad = PI * theta_deg / 180.0;

    printf("\n Degree %lf ---> Radian %lf\n\n",
        theta_deg, theta_rad);

    return 0;
}
```

연습 1.1 데이터 형 변경

위 프로그램에서 사용된 변수 `theta_deg, theta_rad, PI`를 `double`형 대신 `float`형으로 변경해보라.

연습 1.2 라디안을 각도로 변환(rad2deg)

이제 반대로 라디안을 각도로 변환하여 출력하는 프로그램을 작성해보라. 기존의 솔루션에 `rad2deg`라는 이름의 프로젝트를 추가하여 생성하고 프로젝트의 Source Files를 오른쪽 마우스로 클릭하여 `rad2deg.c`이라는 이름의 소스 파일을 새로 만들어 놓고 그 내용 코드를 작성하라.

1.2 좌표 변환

1.2.1 극 좌표 (r,θ)에서 직교 좌표 (x,y)로

$$x = r\cos\theta$$
$$y = r\sin\theta \tag{1.1}$$

이를 C 언어로 표현하면 다음과 같다.

```
x = r * cos(theta);
y = r * sin(theta);
```

C 프로그램에서 수학 함수를 사용하려면 `<math.h>` 헤더 파일이 필요하다. 몇개의 라이브러리 함수는 <C 콘서트, p254> 표에 열거되어 있다. 더 많은 함수들을 보려면 google에서 math.h를 검색하면 된다. 사실 인터넷에는 C에 관한 모든 정보가 실려 있으므로 goole 검색을 하면 정확한 C 언어 정보를 확인할 수 있다.

연습 1.3 극 좌표를 직교 좌표로 변환(polar2rect)
극 좌표의 두 값 (길이, 각도)를 입력 받아 직교 좌표로 출력하는 프로그램을 작성하라.

1.2.2 직교 좌표 (x,y)에서 극 좌표 (r,θ)로

$$r = \sqrt{x^2 + y^2}$$
$$\theta = \tan^{-1}\left(\frac{y}{x}\right) \tag{1.2}$$

이를 C 언어로 표현하면 다음과 같다.

```
r = sqrt(x*x + y*y);
theta = atan(y/x);
```

연습 1.4 직교 좌표를 극 좌표로 변환(rect2polar)
직교 좌표를 입력받아 극 좌표로 변환하는 프로그램을 작성하라.

이때 극 좌표계의 각도는 degree로 표시되도록 하라. 이 프로그램을 이용하여 직교 좌표값 $x = 1$과 $y = 1$을 입력하였을 때 극 좌표값 $r = 1.414214$와 $\theta = 45.0000^\circ$이 출력되는 것을 확인해보라.

연습 1.5 연습 1.2.2 프로그램의 개선

(a) 연습 1.5에서 작성한 프로그램을 이용하여 직교 좌표 $x = 1$과 $y = -1$을 입력하면 $r = 1.414214$와 $\theta = -45.000000^\circ$이 출력되는가 확인해보라.

(b) 연습 1.5 프로그램에 직교 좌표 $x = -1$과 $y = 1$을 입력하면 $r = 1.414214$와 $\theta = 135.000000^\circ$이 출력되는가 확인해보라.

(c) 마지막으로 직교 좌표 $x = -1$과 $y = -1$을 입력하면 $r = 1.414214$와 $\theta = -135.000000^\circ$이 출력되는가 확인해보라. 만약 (b)와 (c)에서 원하는 값이 출력되지 않는다면 무엇이 잘못된 것인가 생각해보고 개선된 프로그램(`rect2polar2`)를 작성하라. (힌트: `atan` 함수는 $-\pi/2$에서 $\pi/2$까지의 범위에서만 동작하므로 `x, y` 값의 부호에 따라 조건문 `if .. else` 조건문을 사용하여야 한다.)

2
정현파 발생

전자시스템은 다양한 형태의 신호들을 이용한다. 그 중 가장 널리 사용되는 것은 정현파이다. 정현파는 오디오 신호를 이루는 기본 신호일 뿐 아니라, 통신시스템을 통해 송수신되는 모든 신호의 기본 형태이기도 하다.

정현파의 일반식은 다음과 같다.

$$x(t) = A\cos(2\pi f_0 t + \phi) \tag{2.1}$$

여기서 A는 진폭, f_0는 주파수 ϕ는 위상변이 즉 $t = 0$일때의 각도이다. 일반적으로 코싸인과 싸인 함수는 그림 2.1에서 보는 것처럼 서로 밀접하게 연관되어 있다. 우리는 정현파를 언급할때 대표적으로 코싸인 함수를 사용한다.

이 장에서는 C 프로그램으로 정현파를 발생하고 저장하는 과정에 대하여 학습한다.

2.1 정현파에 대한 기본 지식

정현파는 각도가 시간에 따라 변하는 삼각함수로 간주된다. 즉 (2)을 다음과 같이 표현할 수 있다.

$$x(t) = A\cos(\underbrace{2\pi f_0 t + \phi}_{\theta(t)}) = A\cos(\theta(t)) \tag{2.2}$$

그림 2.1에서 보는 바와 같이 2차원 공간에서 반지름이 A인 원을 따라 움직이는 점의 궤적을 의미한다. $t = 0$일때 점의 각도가 ϕ에서 출발하여 $2\pi f$의 속도로 원을 따라 움직이면 코싸인은 점 P의 x-축 값의 변화를 의미하며, 싸인은 점 P의 y-축 값의 변화를 의미한다.

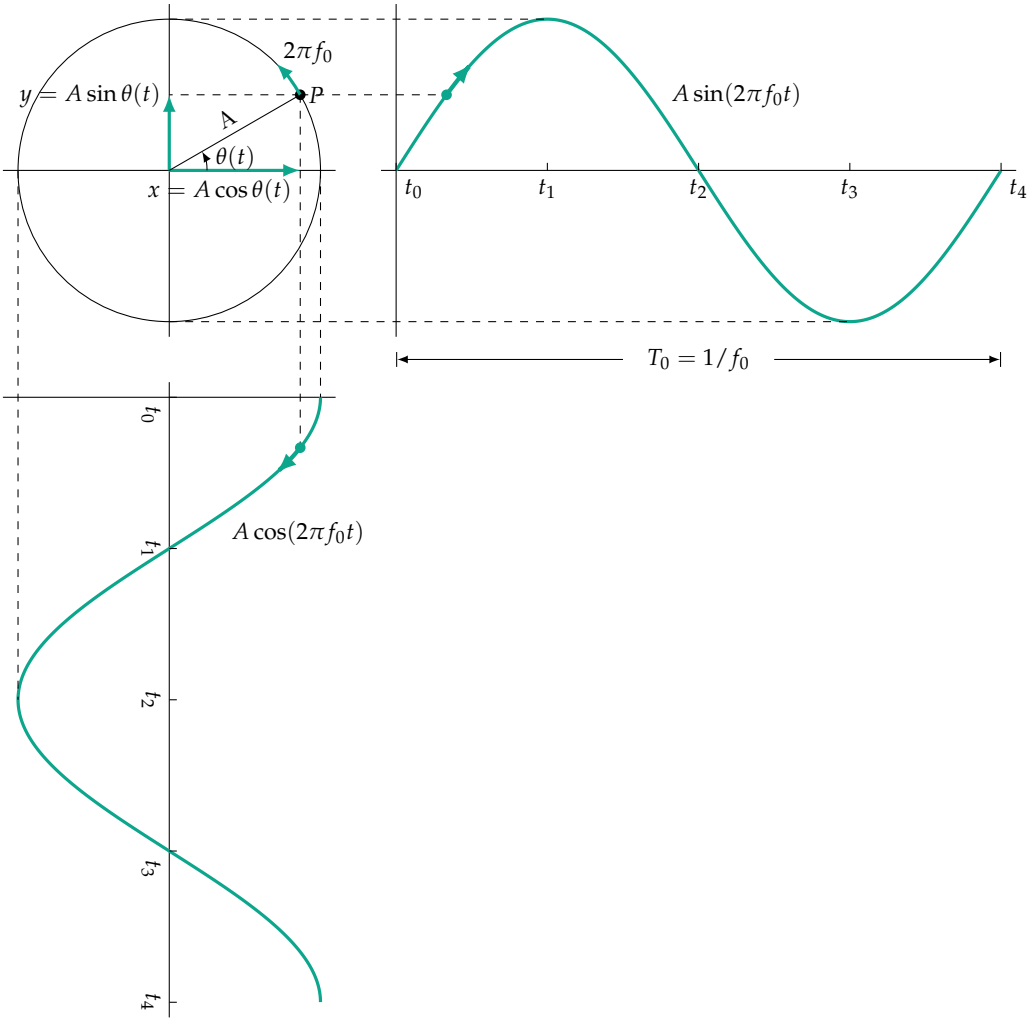


그림 2.1 코싸인과 싸인 정현파

그림 2.1에서 정현파의 진폭은 원의 반지름, 주파수는 1초 동안 원을 회전하는 횟수, ϕ는 시작점에 해당한다.

2.2 컴퓨터에서의 정현파 표현

식 (2.2)의 정현파는 시간과 진폭이 연속적으로 변하는 아날로그 신호이다. 반면에 컴퓨터에서 모든 데이터는 디지털 방식으로 표현된다. 컴퓨터에서 연속 신호를 표현하는 방법은 연속 신호를 일정한 시간 간격 T_s마다 샘플하는 것이다.

이렇게 샘플된 신호는 다음과 같이 표현된다.

$$x(t)\Big|_{t=nT_s} = A\cos(2\pi f_0 t + \phi)\Big|_{t=nT_s} = x(nT_s) \tag{2.3}$$

여기서 T_s는 샘플 주기로 샘플 주파수 F_s의 역수이다. 즉

$$F_s = \frac{1}{T_s}$$

식 (2.3)의 신호는 간단히 $x[n]$으로 표기한다. 이제 샘플링 주파수 F_s를 이용하여 $x[n]$을 표현하면 다음과 같다.

$$\begin{aligned} x[n] &= A\cos(2\pi(f_0/F_s)n + \phi) = A\cos(\hat{\omega}_0 n + \phi) \\ \hat{\omega}_0 &= 2\pi(f_0/F_s) \end{aligned} \tag{2.4}$$

이 수식은 주파수 f_0의 정현파는 매 시간 위상을 $\hat{\omega}_0$만큼씩 증가시킴으로써 만들어질 수 있다.

예를 들어, $f_0 = 1000$ Hz의 정현파를 $F_s = 8000$ Hz로 샘플링하면 ($A = 10$, $\phi = \pi/3$을 가정하면) 다음과 같이 표현될 수 있다.

$$x_1[n] = 10\cos(2\pi(1000/8000)n + \pi/3) = 10\cos(0.25\pi n + \pi/3)$$

즉 샘플링된 이산 신호의 디지털 각속도는 $\hat{\omega}_0 = 0.25\pi$이다. 그러므로 위상은 $n = 0$일때 초기값이 $\pi/3$이며, n이 증가함에 따라 0.25π 라디안 만큼 씩 증가한다.

2.3 프로그램 과제

2.3.1 코싸인 함수의 각도를 20° 씩 증가시키면서 출력

각도가 0°에서 360°까지 20° 씩 변할때 코싸인 함수값을 출력하려고 한다.

$$x[n] = \cos(\theta(n)).$$

여기서 $\theta(n)$은 시간 인덱스 n에서의 각도를 의미한다. 이제 각도를 20° 씩 증가한다면 시간 인덱스 n에서 각도는 라디안으로 다음과

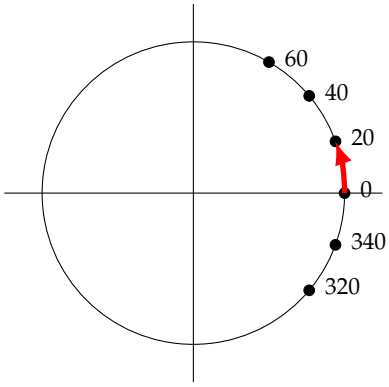


그림 2.2 일정 스텝 간격으로 증가하는 각도

같이 표현될 수 있다.

$$\theta(n) = 20 \times \pi \times n/180.$$

이를 for 반복문을 이용한 C 프로그램으로 다음과 같이 구현할 수 있다.

```
for(n=0; n<18; n++)
    x[n] = cos(20*PI*n/180.0);
```

그러나 이 프로그램은 for 루프 안에서 동일한 곱셈 계산을 반복하는 단점이 있다. 곱셈 계산은 덧셈 계산에 비해 복잡하며 시간이 많이 소요된다.

이제 $\theta(n)$ 값이 매번 일정한 값 만큼 증가한다는 점을 이용하여 프로그램을 작성할 수 있다.

$$\theta(n+1) = \theta(n) + 20 \times \pi/180.$$

```
delta = 20.0 * PI /180.0; // radian <- 20 degree
theta = 0.0;
for(n=0; n<18; n++) {
    x[n] = cos(theta);
    theta += delta;
}
```

전체 프로그램을 완성하고 실행시켜 보아라.

연습 2.1 한주기 이상 길이의 코싸인 함수 발생

위 프로그램을 개조하여 코싸인 함수를 10 주기 정도 발생시키는 프로그램을 만들려고 한다. 위 프로그램에서 `n`을 180으로 변경하면 10 주기 길이의 코싸인 함수가 발생될 것이다. 그러나 이 경우 `theta`는 계속 증가한다. 경우에 따라서는 이 값이 너무 커져서 문제가 발생할 가능성도 있다. 사실 `theta`의 값은 항상 0에서 2π 범위 안에 있으므로, 이 범위를 벗어나는 경우 `theta`를 조정하므로써 다음과 같이 문제를 해결할 수 있다.

```
if (theta>TWOPI), theta -= TWOPI;
```

여기서 `TWOPI`는 `2*PI`로 정의되어 있어야 한다.

2.3.2 특정 주파수를 가진 정현파 발생

식 (2.4)을 이용하면 주파수 f_0의 정현파를 초당 F_s개 발생시킬 수 있다. 여기서 초기 위상 ϕ는 0으로 정하고, 진폭은 $A = 10$, 주파수 $f_0 = 1000$ Hz, 샘플링 주파수는 $F_s = 8000$ Hz로 설정하면 정현파의 식은 다음과 같이 표현된다.

$$x[n] = 10\cos(2\pi(1000/8000)n) = 10\cos(0.25\pi n) \tag{2.5}$$

연습 2.2

[정현파 발생하는 프로그램 작성(gen_tone)] 신호 (2.5)를 3초 길이 만큼 발생시키는 프로그램을 작성하여라. 먼저 3 초 동안 몇 개의 정현파 샘플이 있을지를 계산해보라.

연습 2.3 정현파 발생을 위한 함수 작성

앞의 예제에서 작성한 프로그램을 다음과 같이 함수화하라.

```
void gen_tone(double A, double f0, double phi, double
    fs, double dur, double *x)
```

2.3.3 소리를 파일로 저장하기

일반적으로 C 프로그램에서 데이터를 저장하는 방법에는 파일의 유형에 따라 다음과 같이 크게 두 가지로 구분된다.

- ASCII(아스키) 파일로 저장
- Binary(이진) 파일로 저장

ASCII 파일은 우리가 메모장 같은 프로그램으로 쉽게 열어서 내용을 확인할 수 있는 ASCII 문자로 된 파일을 의미하며, binary 파일은 이진수로 되어 있으므로 특별한 프로그램으로만 내용을 확인할 수 있다.

ASCII 파일은 `printf`에서 사용했던 것과 같이 형식에 따라 파일에 기록한다. (상세한 내용은 C 언어 교재의 파일 입출력 부분을 참고하라)

이제 정현파 신호를 시간 `n`과 정현파 신호 값 `x[n]`을 쌍으로 한 줄씩 저장하려면 다음의 세 단계를 거친다.

1. 파일을 연다.

```
FILE *fp;
...
fp = fopen("tone.dat", "w");
```

2. 데이터를 파일에 저장한다. 아스키 파일에서는 데이터를 한번에 한 셋트씩 기록하게 되므로 `for` 루프를 통해 여러 셋트의 데이터를 만들어 내는 경우 `for` 루프 안에서 데이터를 파일에 기록해야만 한다.

```
fprintf(fp, "%d %lf\n", n, x[n]);
```

3. 파일을 닫는다.

```
fclose(fp);
```

연습 2.4 소리 데이터를 아스키 파일로 저장하기(save_tone_acsii)
이 방식을 이용하여 예제 2.2 프로그램에 추가하여 소리 데이터를 파일로 저장하고 파일을 메모장으로 열어서 데이터가 내가 원하는 식으로 저장되어 있는지 확인해보라. 데이터는 visual studio의 해당 프로젝트 폴더 안에 저장되어 있다.

2.3.4 소리를 PCM 파일로 저장하기

소리를 컴퓨터 파일로 저장하는 방법 중 하나는 PCM 파일로 저장하는 것이다. 일반적으로 소리를 저장하는 PCM 포맷은 각 샘플을 16 비트 2의 보수 이진수로 표현한다. C 언어에서 16 비트 2의 보수는 `short`로 표현된다. 그러므로 C에서 `float`나 `double`로 표현되는 정현파는 PCM 파일에 저장되기 위해 `short`형으로 변환될 필요가 있다. C 언어에서 PCM 형식으로 저장하려면 다음의 사항을 고려해야 한다.

- 정현파의 진폭 A를 최대 1을 기준으로 한 상대적인 값으로 결정한다.
- 16 비트 2의 보수의 범위는

$$-32768 = -2^{15} \leq x \leq 2^{15} - 1 = 32767$$

이므로 다음과 같이 `short`형으로 변환해준다.

```
#define PI 3.1415926535
#define TWOPI (2.0 * PI)
#define MAXAMP 32768
...
double x[N];// range ( -1.0 ~ +1.0)
short xs[N];
...
A = 0.2;
theta = 0.0;
del = TWOPI * (f0 / fs)
for (n=0; n<N; n++) {
   x[n] = A * cos(theta);
```

```
    theta = del;
    if (theta > TWOPI), theta -= TWOPI;
    xs[n] = (short) round(MAXAMP * x[n]);
}
```

예를 들어, short 형의 xs[n] 데이터 N 개를 MyPCM.pcm에 저장하려면 다음과 같이 하면 된다(C 언어 교재의 파일 입출력 참고).

```
FILE *fp;
...
fp = fopen("MyPCM.pcm", "wb");
fwrite(xs, sizeof(short), N, fp);
fclose(fp);
```

연습 2.5 소리를 PCM 파일로 저장하기 (pcm_write)

최대 진폭이 0.3, 주파수 1000 Hz, 샘플링 주파수 8000 Hz, 샘플당 16 비트의 신호를 PCM 파일로 저장하고 제대로 저장되었는지 확인해보라.

연습 2.6 PCM 파일로 저장하는 함수 (pcm_write) 작성

다음과 같이 정의되는 함수를 작성하고, 예제 2.5와 동일한 신호를 저장해보라.

```
pcm_write(short *x, int ch, int len, FILE *fp);
```

2.3.5 소리를 wav 파일로 저장하기

wav 파일은 컴퓨터를 이용하여 소리를 저장하고 재생하기 위한 표준 파일 형식이다. wav 파일 형식은 일종의 binary 파일로서 PCM 방식으로 데이터를 저장하며, 파일의 머리 부분에 데이터의 샘플링 주파수, 채널 수, 샘플 당 비트 수, 전체 데이터의 샘플 수 등의 정보를 담고있는 44 바이트 길이의 헤더(header)를 가진다. 이 교재에서는 wav 파일을 저장하기 위한 용도로 간단한 wav 파일 저장 함수 wav_write가 제공된다. 이 함수의 용법은 다음과 같다.

```
wav_write(short *x, int ch, int fs, int len, FILE *fp);
```

예를들면, 샘플링 주파수 8000, 채널 수 1, 길이 3초의 신호 x를 저장하려면 다음과 같이 표현한다.

```
wav_write(x, 1, 8000, 24000, FILE *fp);
```

이제 주파수 $f_0 = 1000\,\mathrm{Hz}$, 진폭 7500, 길이 3초, 주파수 $F_s = 8000$로 샘플링된 단일 채널 신호를 만들어서 저장하는 프로그램을 작성하고 그 소리를 들어서 확인해보라.

2.3.6 비트 음 발생

주파수가 각각 f_1과 f_2인 두 정현파의 합은 삼각함수의 특성을 이용하면 두 정현파의 곱으로 표현될 수 있다.

$$\begin{aligned} x[n] &= \cos(2\pi(f_1/F_s)n) + \cos(2\pi(f_2/F_s)n) \\ &= 2\cos(2\pi(\Delta f/F_s)n) \times \cos(2\pi(f_c/F_s)n) \end{aligned} \tag{2.6}$$

여기서 $\Delta f = |f_2 - f_1|$이고 $f_c = (f_1 + f_2)/2$이다. Δf가 아주 작은 값이면, 상대적으로 높은 주파수의 $\cos(2\pi(f_c/F_s)n)$ 신호의 진폭이 낮은 주파수의 $\cos(2\pi(\Delta f/F_s)n)$ 신호에 의해 변조되는 현상이 발생한다. 이러한 현상은 진폭변조(AM)이라고 하며, 가청 주파수 영역에서 이러한 현상이 발생하는 것을 비트(beat)음이라고 부른다. $F_s = 8000$, $f_c = 1000$, $\Delta f = 10$으로 설정하고 5 초간 비트음을 발생하여 wav 파일로 저장한 다음 그 소리를 들어보라. 프로그램을 작성할 때는 두 정현파의 곱을 사용하지말고 합의 공식을 사용하도록 하라. Δf를 서서히 증가시키면서 언제까지 비트 음이 들리는지 확인해보라.

2.3.7 피아노 음계의 한 옥타브에 해당하는 정현파 발생

피아노 건반의 한 옥타브(octave)는 그림 2.3에서 보는 바와 같이 12개의 흑색과 백색 건반으로 구성되어 있다. 건반에 적혀있는 번호는 MIDI 번호로서 건반의 음을 번호화한 것이다. MIDI 번호 하나는 반음 차이로 한 옥타브는 12개의 반음으로 구성된다. 한 옥타브는 주파수가 2배 되는 구간을 의미하며, 반음이란 두 주파수의 차가 아닌 비율을 의미한다. 따라서 반음에 해당하는 등비 r은 다음과 같이

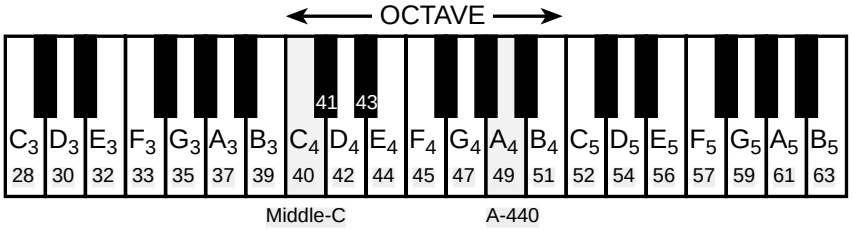


그림 2.3 피아노 건반의 가운데 세 옥타브에 대한 MIDI 번호

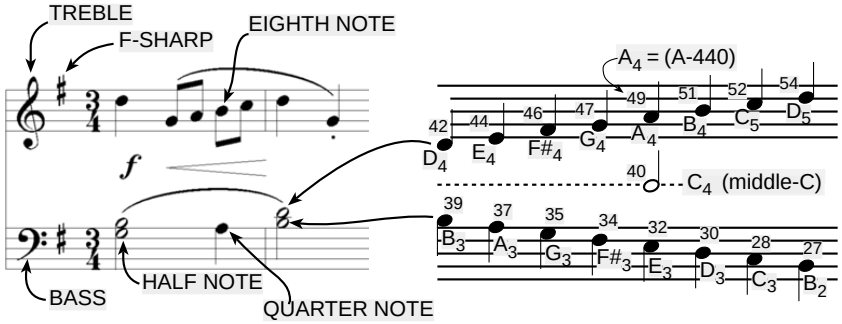


그림 2.4 음계

구해진다.

$$r^{12} = 2 \qquad \Longrightarrow \qquad r = 2^{1/12} = 1.0595 \tag{2.7}$$

그림 2.3의 건반에 적힌 A_4는 4번째 옥타브의 라(A)음을 의미하며 MIDI 번호는 49번이다. A_4는 정수값의 주파수 440 Hz를 갖기 때문에 악기의 조율에 기준으로 사용되기도 한다. 식 (2.7)를 이용하면 MIDI 번호 40 번의 낮은 도(C_4)음의 주파수는 다음과 같이 계산될 수 있다.

$$f_{C_4} = 440 \times 2^{(40-49)/12} \approx 261.6 \text{ Hz}$$

이러한 방식으로 다장조 음계의 각 음에 대한 주파수를 계산하면 다음 표와 같다.

그림 2.4에서 설명한 바와 같이, 악보 상의 각 음표는 음 높이와 음의 길이에 의해 규정된다. 예를 들어, ♩ = 80은 4분음표가 분당 80번 반복되는 것을 의미하므로 4분음표의 하나의 길이는 $60/80 \approx 0.75$초가 된다.

이제 다장조 음계를 연주하는 프로그램(`play_octave`)을 작성해

표 2.1 다장조 음계의 MIDI 번호와 주파수

음	C_4	D_4	E_4	F_4	G_4	A_4	B_4	C_5
MIDI 번호	40	42	44	45	47	49	51	52
주파수(Hz)	262	294	330	349	392	440	494	523

보자. 먼저 연주할 음의 주파수를 직접 사용하는 대신 다음과 같이 MIDI 번호와 A_4음의 주파수를 이용하여 계산하도록 한다.

```
double fk;
int note_number[] = {40, 42, 44, 45, 47, 49, 51, 52};
int k;
...
for (k=0; k<8; k++) {
    fk = 440.0 * pow(2.0,(note_number[k]-49)/12.0);
    /* generate nsamples of sinusoid of frequency fk */
    ...
}
```

샘플링 주파수 $F_s = 11025$, 진폭 $A = 7500$, 신호는 short 데이터형을 사용하는 것으로 가정한다. 각 음이 0.4초 씩 지속되는 것으로 설정할 경우, 한 음의 샘플 갯수는 $0.4 \times 11025 = 4410$개이다. 이 길이의 배열은 다음과 같이 미리 준비해 둘 수 있다.

```
short x[4410];
```

그러나 어떤 경우 음마다 길이가 달라질 수 있기 때문에, 배열을 사용하는 대신 다음과 같이 동적 메모리 할당을 이용할 수도 있다.

```
short *x;
x = (short *) calloc(nsamples, sizeof(short));
```

calloc 대신 malloc를 사용할 수 도 있다.

```
x = (short *) malloc(nsamples*sizeof(short));
```

calloc는 모든 배열 값을 0으로 초기화하지만, malloc는 초기화하지 않는다는 점이 다르다. 동적 메모리 할당을 한 경우 할당해준 메모리를 더 이상 사용하지 않으면 메모리를 해제시켜 주어야 한다.

```
free(x);
```

연습 2.7 각 음에 대한 신호 발생을 위한 함수 작성(write_note)

크기 A, 건반 번호 key, 음의 지속 시간 dur을 갖는 음을 FILE *fp에 기록하는 함수를 작성하고, 이 함수를 이용하여 다장조 음계 소리를 octave.pcm에 저장하는 프로그램을 완성하라.

```
void write_note(double A, int key, double dur,
FILE *fp);
```

2.4 프로그램 과제: 컴퓨터를 이용한 악보 연주

이 과제에서는 그림 2.4에 설명된 바와 같은 음 높이와 길이 표시를 이용하여 간단한 음악을 연주하는 프로그램을 작성해본다. 컴퓨터를 이용한 음악 연주에서 악보는 MIDI 형태와 유사한 것으로 제공된다. 즉, 원래의 악보로부터 MIDI 형태의 데이터를 추출한 다음 데이터 파일의 형태로 컴퓨터 프로그램에 입력된다. 이 MIDI 악보에는 각 음의 높이가 MIDI 번호의 형태로 주어지며 음의 길이 정보도 담겨있다. 쉼표의 경우 음 대신 쉼표 길이 만큼의 묵음이 대신된다.

실제 음악처럼 연주하려면 여러 가지 개선들이 더 필요하다. 여기서는 간단히 두 가지 개선 만을 고려한다.

2.4.1 하모닉 성분을 포함하기

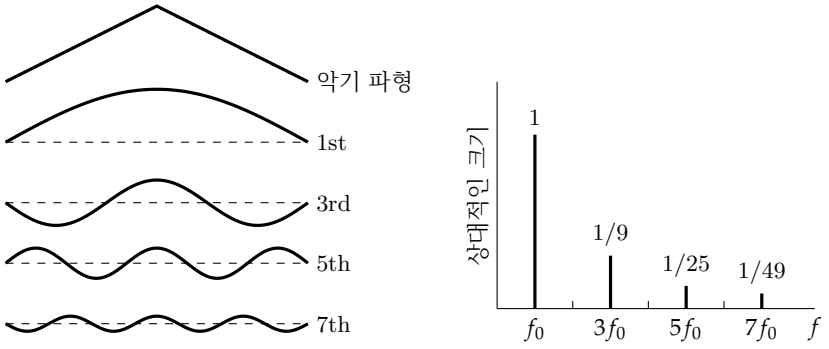


그림 2.5 현악기의 하모닉 성분들의 크기와 위상 변위

악기에 따라서 음색이 다른 이유는 동일한 높이의 음이라 하더라도 스펙트럼의 모양이 다르기 때문이다. 예를 들면, A4음의 기본 주파수는 440 Hz이지만 악기에 따라 기본 주파수의 정수배 음들(하모닉)이 포함된다. 그리고 이 하모닉 성분들의 크기는 악기에 따라 또 다르다. 하모닉 성분들이 존재하므로 음이 풍성하게 들리고 악기 고유의 음색도 내게 된다.

그림 2.5은 현악기에 포함된 하모닉 성분들의 한 예이다. 현악기를 활로 연주할 때 현악기의 줄은 삼각형 모양으로 변형이 되면서 진동하게 된다. 이 삼각형 모양의 파형은 기본 주파수와 기본 주파수의 홀수배 주파수를 갖는 정현파의 합으로 구성되어 있다. 하모닉 성분들은 기본 주파수 성분에 비하여 상대적으로 작은 크기를 진폭을

갖으며 위상 변위도 다르다. 즉 3번째 하모닉 성분은 기본 주파수 성분에 비해 상대적인 크기는 1/9이며 위상은 180° 뒤집혀있다.

2.4.2 진폭의 변화를 고려하기

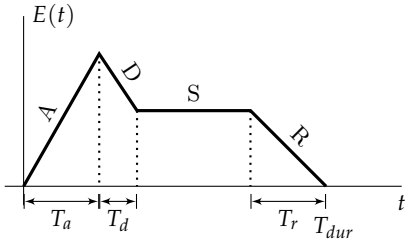


그림 2.6 음 세기 변화에 대한 ADSR 모형

실제로 음악의 각 음들은 동일한 진폭을 가지지 않는다. 사람의 손이 피아노 건반을 두드릴때 처음에는 큰 소리가 나다가 서서히 소리가 소멸하기 마련인데, 이러한 변화는 그림 2.6과 같은 ADSR(Attack-Decay-Sustain-Release) 모형으로 설명된다.

Attack 구간은 손이 건반을 두드릴때 진폭이 짧은 시간 동안(T_a) 최대 진폭 A_{max}까지 선형적으로 증가하는 구간으로 다음과 같이 표현된다.

$$E(t) = A_{max} \times \frac{t}{T_a} \tag{2.8}$$

Decay 구간은 T_d 시간 안에 최대 값에서 일정한 수준 A_{sus}으로 감소하는 구간으로 다음과 같이 표현된다.

$$E(t) = \frac{A_{sus} - A_{max}}{T_d} \times (t - T_a) + A_{max} \tag{2.9}$$

Sustain 구간은 진폭이 일정한 수준 A_{sus}으로 유지되는 구간이다. S 구간은 별도로 구간 시간 대한 규정을 두지 않으며 전체 구간 T_{dur}에서 $T_a + T_d + T_r$을 뺀 나머지 시간으로 정의된다. 이렇게 함으로써 S 구간은 다른 구간에 의해 가변적으로 조정될 수 있다.

$$E(t) = A_{sus} \tag{2.10}$$

마지막으로 Release 구간은 손이 완전히 악기를 놓음으로써 T_r 시간

동안 음이 소멸되는 구간으로 다음과 같이 표현된다.

$$E(t) = -\frac{A_{sus}}{T_r} \times (t - (T_{dur} - T_r)) + A_{sus} \tag{2.11}$$

프로그램은 연속 시간 t가 아닌 이산 시간 nT_s를 이용하므로 위의 식들은 표본화된 값으로 변환해 주어야 한다. 예를들면 Attack의 경우 식은 다음과 같이 변환 될 수 있다.

$$E[n] = A_{max} \times \frac{n}{N_a} \tag{2.12}$$

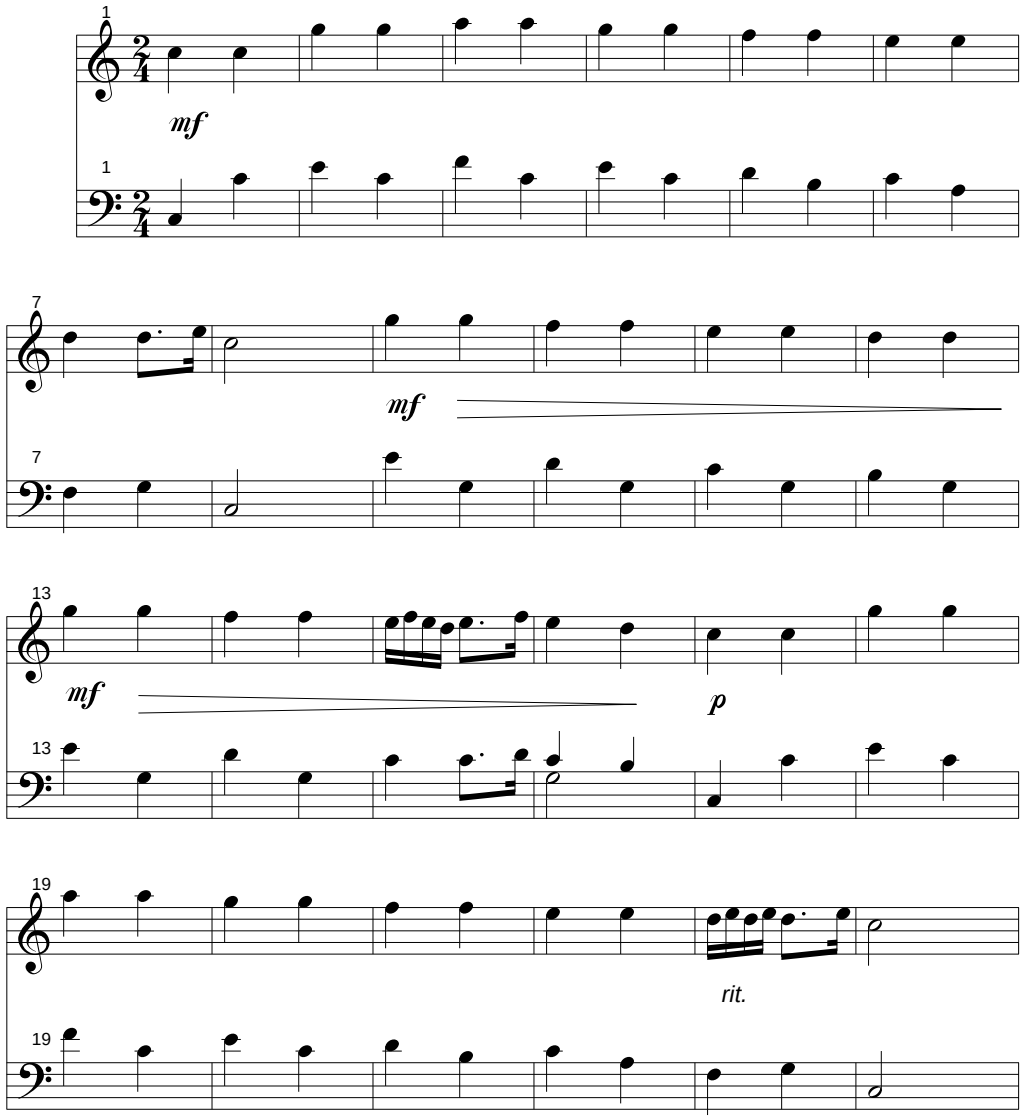
여기서

$$N_a = \text{round}(T_a / f_s).$$

위의 두 가지 개선안을 이용하여 다음 악보의 음악을 연주하는 프로그램을 작성해보자. 우리는 악보로부터 두 가지 정보를 추출하여 악보 파일로 저장한 다음 이를 프로그램에서 읽어서 소리 파일로 저장하면 된다.

Twinkle, Twinkle Little Star

W. A. Mozart



3
콘볼루션

콘볼루션은 유한 길이 필터(FIR: finite impulse response)에 적용되는 연산으로, 필터 입력$x[n]$과 필터의 임펄스 응답(필터 계수)$h[n]$으로부터 필터출력$y[n]$은 다음과 같이 표현된다.

$$y[n] = h[n] * x[n] = \sum_{k=0}^{M} h[k]x[n-k] \tag{3.1}$$

콘볼루션을 C 프로그램으로 구현하는 방법은 다양하지만, 여기서는 두가지 방법을 소개한다.

3.1 콘볼루션의 직접 계산 방식

식 (3.1)은 그림 3.1과 같은 블록 다이어그램으로 표현된다. 여기서 입력 신호는 지연소자를 지나면서 각각의 필터계수 $h[k]$와 곱해지며

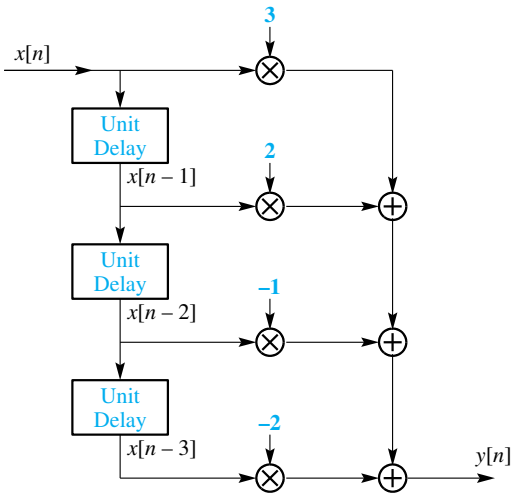


그림 3.1 필터 계수 $h[n]$는 $\{3,2,1,-2\}$인 FIR 필터의 직접 구현 방식.

그 결과들은 합쳐져서 출력 $y[n]$을 이루게 된다. 이제 지연된 입력 신호의 벡터를 w[k]로 표현하면 그림 3.1은 다음과 같이 구현될 수 있다.

```
for (n=0; n<Lx; n++) {
   /* shift input delay */
   for (k=Lh-1; k>0; k--)
      w[k] = w[k-1];
   w[0] = x[n];
   /* multiply with filter coefficients and sum up */
   y[n] = 0.0;
   for (k=0; k<Lh; k++)
      y[n] += h[k] * w[k];
}
```

위의 프로그램은 다음과 같이 함수로 표현할 수 있다.

```
void fir_direct(double *x, double *w, double *h, int Lh,
    double *y) {
   int n, k;

   for (n=0; n<Lx; n++) {
      for (k=Lh-1; k>0; k--)
         w[k] = w[k-1];
      w[0] = in[n];
      y[n] = 0.0;
      for (k=0; k<Lh; k++)
         y[n] += h[k] * w[k];
   }
}
```

한가지 고려해야하는 사항은 필터의 출력 $y[n]$의 길이 L_y가 입력 $x[n]$의 길이 L_x보다 정확하게 $h[n]$의 길이에서 하나를 뺀 만큼 길다는 것이다.

$$L_y = L_x + L_h - 1$$

연습 3.1 fir_direct 함수를 이용한 콘볼루션 계산 시험

`fir_direct` 함수를 이용하여 입력 {2.0, 4.0, 6.0, 4.0, 2.0}과 필터 {1.0, -1.0, 2.0}에 대한 계산을 수행하는 프로그램을 작성하고 출력이 {2.0, 2.0, 6.0, 6.0, 10.0, 6.0, 4.0}인지 확인해보라.

3.2 순환 버퍼를 이용한 콘볼루션 구현

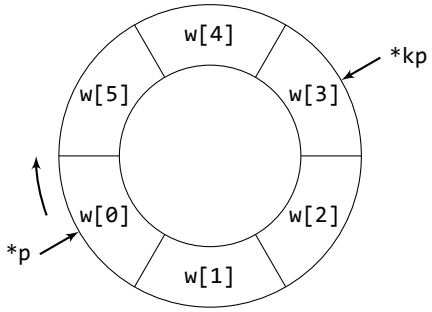


그림 3.2 길이가 6인 순환 버퍼의 구조

3.1 절에서 소개한 직접 구현 방식의 단점은 새로운 입력이 인가될 때 마다 지연 버퍼 `w[n]`의 모든 값들을 한자리씩 이동시켜야 한다는 것이다. 이 문제점은 순환 버퍼를 이용하여 쉽게 해결될 수 있다. 그림 3.2는 길이가 6인 순환 버퍼의 구조이다. 포인터 `*p`는 순환 버퍼를 시계방향으로 돌면서 입력을 저장한다. 포인터 `*p`가 가르키는 곳이 새로운 입력 샘플이 저장되는 곳이며 지연된 입력들은 반시계 방향으로 저장되어 있으며 `kp`에 의해 지정된다. 이를 이용한 콘볼루션 함수는 `fir` 함수와 같다.

위 프로그램에서 `*p`는 순환 버퍼에서 입력을 받는 주소로 처음에는 0으로 초기화되며 계속 입력이 들어옴에 따라 시계방향으로 이동한다(즉 주소가 하나씩 증가한다.). 주소가 이동하여 `*p=Lh`이 되면 다시 0으로 리셋시킨다. 이 프로그램에서 `*sig`는 입력과 출력에 동시에 사용된다는 점이 특이하다.

```
void fir(double *sig, double *w, double *h, int Lx, int
    Lh, int *p) {
   double out=0.f;
   int kp;//index for circular buffer

   for (n=0; n<Lx; n++) {
      w[*p] = sig[n];//accept new input
      *p = (*p != Lh-1 ? *p+1 : 0);//check circular
      for (k=0; k<Lh; k++) {
         kp = *p + k;
         kp = (k<Lh ? kp : kp-Lh);//check circular
         out += (w[kp]*h[Lh-1-k]);
      }
      sig[n] = out;//output filter output
      out = 0.f;
   }
}
```

4

DTMF 신호 인코딩과 디코딩

이 과에서는 전화기에서 번호를 누를때 번호 정보를 전달하는 DTMF(Dual Tone Multiple Frequency) 터치-톤 다이얼링을 소개한다. 번호 정보는 두개의 서로 다른 주파수의 정현파로 부호화된 FSK(Frequency Shift Keying)의 형태로 송신된다. 수신기에서는 번호 별로 정해진 대역통과 필터를 통과시킨 다음 각각의 필터 출력을 비교한 결과로 판정된다.

4.1 DTMF 배경 지식

터치-톤 다이얼링에서 이용하는 DTMF는 전화기의 다이얼 번호 마다 두 개의 주파수가 배정되어, 번호를 누르면 두 정현파가 발생된다. 그림 4.1는 전화기의 다이얼 패드 번호에 배정된 주파수이다. 다이얼 패드의 번호는 마치 4×3 행렬처럼 배치되어 있다. 각 번호에는 행의 주파수와 열의 주파수가 하나씩 배정되어 있다. 예로, 번호 4를 누르면 770 Hz의 정현파와 1209 Hz의 정현파가 동시에 발생되어 합쳐진다. DTMF에서 사용되는 주파수들은 서로 합과 차의 주파수들과 연관되지 않으며 상호간에 정수배가 되지않도록 주의깊게 선택되었다. 이는 DTMF 신호가 비선형 소자를 통과할때 발생할 수 있는 이미지 주파수들로 인해 신호 검파에 문제가 발생하기 않도록 하기 위함이다.

4.2 DTMF 신호 발생

DTMF 신호는 $f_s = 8000$ Hz의 주파수로 샘플링된다고 가정한다. 행과 열의 주파수를 각각 f_r과 f_c이라고 하면 DTMF 신호는 다음과

표 4.1 DTMF 키패드의 주파수(Hz)

freq	1209	1336	1477
697	1	2	3
770	4	5	6
852	7	8	9
941	*	0	#

같이 표현된다.

$$x[n] = A\cos(\hat{\omega}_r n) + A\cos(\hat{\omega}_c n)$$

여기서

$$\hat{\omega}_r = 2\pi\left(\frac{f_r}{f_s}\right), \qquad \hat{\omega}_c = 2\pi\left(\frac{f_c}{f_s}\right)$$

진폭 A는 신호의 진폭이 `short` 데이터의 최대값인 $2^{15} = 32768$를 넘지 않도록 설정되어야 한다. 여기서는 $A = 10000$으로 정한다. 실생활에서 정현파의 지속 시간과 번호 사이의 쉼 구간은 가변적이지만, 여기서는 정현파의 지속 시간은 0.2초 쉼 구간은 0.05초로 정한다.

일련의 번호 `phone_number`에 대하여 DTMF 신호를 발생하여 파일 포인터 `fp`가 지정하는 pcm 파일로 저장하는 함수 `dtmfdial`을 완성하여라.

```
void dtmfdial(char *phone_number, FILE *fp)
{
    char keypad[] = {"123456789*0#"};
    double freqRow[] = {697.0, 770.0, 852.0, 941.0};
    double freqCol[] = {1209.0, 1336.0, 1477.0};
    ...
}
```

여기서 `phone_number`는 전화번호의 문자열이다. C 라이브러리 함수 `strchr`을 이용하여 전화번호의 처음 번호부터 하나씩 `keypad`의 문자와 비교하여 번호가 있는 열과 행을 확인하고 해당 주파수를 이용하여 신호를 발생하는 방법을 사용한다. C 언어의 문자열 함수를 참고하여 적절하게 사용할 수 있는 것이 무엇인지 파악해보라.

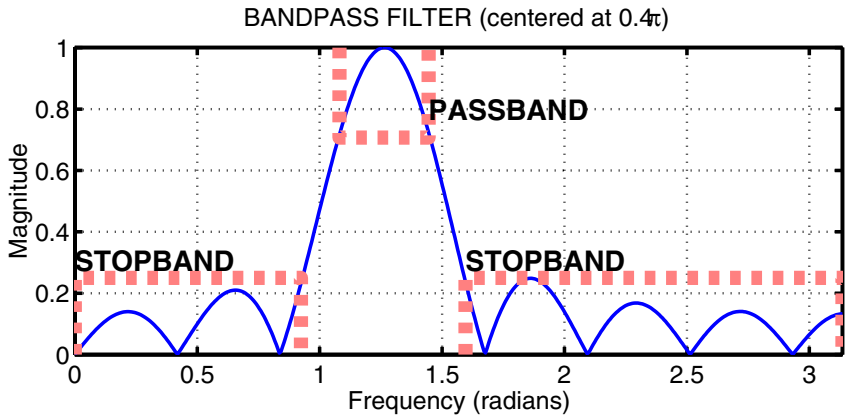


그림 4.1 대역통과 필터의 주파수 응답과 통과대역 및 차단대역

4.3 DTMF 신호 검파

DTMF 신호 검파는 수신된 신호의 두 정현파 주파수를 알아냄으로써 이루어진다. 그림 4.1에서 보는 바와 같이 DTMF는 4개의 행의 주파수 중에서 하나는 판별하고 3개의 열의 주파수 중의 하나를 판별하면 된다. 이들 주파수를 확인하려면 수신된 신호를 그림 ??와 같이 행과 열의 주파수에 따른 대역통과 필터뱅크를 통과하여 각각의 필터뱅크에서 신호의 크기가 가장 큰 것을 선택하면 된다.

간단한 대역통과 필터의 임펄스 응답은 다음과 같이 주어진다.

$$h[n] = \beta \cos(\hat{\omega}_c n), \quad 0 \le n < L \tag{4.1}$$

$\hat{\omega}_c$는 통과대역의 중심 주파수이다. 예를들면, $f_c = 697$Hz의 신호에 대한 통과필터의 경우 $\hat{\omega}_c = 2\pi 697/8000$이므로 필터의 중심은 0.174π에 놓여지게 된다.

필터 길이 $L = 51$로 정한다. 이때 주파수 응답의 최대 값이 1이 되도록 $\beta = 1/L$로 결정한다. 일반적으로 대역통과 필터의 대역폭은 그림 4.1 최대값에서 크기가 $1/\sqrt{2}$가 되는 주파수 범위이다.

입력된 DTMF 신호에 대하여 대역통과 필터들의 출력을 계산하는 과정은 다음과 같다.

```
for (i=0; i<4; i)
   fir_bpf(x, h, L,
   for (j=0; j<3; j++)
```