중위수 필터 구현

문제: “float find_median(float dist) { }”: dist를 입력받아 최근 N개의 dist 중에서 중간값을 반환하는 함수 구현하기

최근 N개의 값을 유지하기 위한 배열, 중간값을 찾기 위해 정렬된 배열, 총 2개의 배열을 사용한다.

1. 최근 N개의 값만 저장하기: 크기가 N인 배열을 생성하고, “index = (index + 1) % N;“으로 원 형태의 배열을 구현한다. 0 ~ N-1 인덱스를 사용하며 인덱스가 N이 되면 0이 되어 자연스럽게 N 번째 전에 입력된 값을 덮어쓴다.
2. 중간값 찾기: 중간값을 찾기 위해선 N개의 데이터를 정렬해야 한다. 데이터가 하나씩 삭제되고 추가되기 때문에 삽입정렬을 사용하고, 데이터를 삽입할 위치를 찾기 위해 이분탐색을 한다. 데이터를 삭제하거나 추가할 때마다 배열을 한 칸씩 뒤로 밀거나 앞으로 당겨야 되기 때문에 O(N) 시간 복잡도를 가진다.


```
// Arduino pin assignment
#define PIN_LED  9
#define PIN_TRIG 12
#define PIN_ECHO 13

// configurable parameters
#define SND_VEL 346.0     // sound velocity at 24 celsius degree (unit: m/sec)
#define INTERVAL 25       // sampling interval (unit: msec)
#define PULSE_DURATION 10 // ultra-sound Pulse Duration (unit: usec)
#define _DIST_MIN 100     // minimum distance to be measured (unit: mm)
#define _DIST_MAX 300     // maximum distance to be measured (unit: mm)

#define TIMEOUT ((INTERVAL / 2) * 1000.0) // maximum echo waiting time (unit: usec)
#define SCALE (0.001 * 0.5 * SND_VEL)     // coefficent to convert duration to distance

#define _EMA_ALPHA 0.5    // EMA weight of new sample (range: 0 to 1)
                          // Setting EMA to 1 effectively disables EMA filter.

#define MIN 0
//#define MAX (SND_VEL * SCALE * TIMEOUT + 1) // unit: mm
#define MAX 32000 // unit: mm
#define N 30

// global variables
unsigned long last_sampling_time;   // unit: msec
float dist_prev = _DIST_MAX;        // Distance last-measured
float dist_ema;                     // EMA distance
float arr_dist[N+2];
float arr_sorted[N+2];
int id = 0, cnt = 0;

void setup() {
  // initialize GPIO pins
  pinMode(PIN_LED,OUTPUT);
```

// 상수 선언부

MAX: 오름차순 정렬된 배열에서 값을 지울 때마다 맨 뒤에 임시로 넣어주는 값이다. 이분탐색이 정상적으로 동작하게 한다.
N: 중간값을 구할 때 이용할 값의 개수.

// 변수 선언부
float arr_dist[N+2]: 최근 N개의 값들을 입력받은 순서대로 저장하는 배열.
float arr_sorted[N+2]: arr_dist[]의 값들을 오름차순 정렬한 배열.
id: arr_dist[]의 현재 값을 저장할 인덱스.
cnt: 현재까지 입력된 값의 개수, min(cnt, N)을 저장함.

```
  pinMode(PIN_TRIG,OUTPUT);
  pinMode(PIN_ECHO,INPUT);
  digitalWrite(PIN_TRIG, LOW);
  digitalWrite(PIN_LED, 1);

  // initialize serial port
  Serial.begin(57600);
  while(!Serial);
}

void loop() {
  float dist_raw, dist_filtered, dist_median;

  // wait until next sampling time.
  // millis() returns the number of milliseconds since the program started.
  // will overflow after 50 days.
  if (millis() < last_sampling_time + INTERVAL)
    return;

  // get a distance reading from the USS
  dist_raw = USS_measure(PIN_TRIG,PIN_ECHO);

  // Modify the below if-else statement to implement the range filter
  if ((dist_raw == 0.0) || (dist_raw > _DIST_MAX)) {
      dist_filtered = dist_prev;
  } else if (dist_raw < _DIST_MIN) {
      dist_filtered = dist_prev;
  } else {    // In desired Range
      dist_filtered = dist_raw;
      dist_prev = dist_raw;
  }
  dist_ema = _EMA_ALPHA * dist_filtered + (1 - _EMA_ALPHA) * dist_ema;
  dist_median = find_median(dist_raw);

  // output the read value to the serial port
  Serial.print("Min:");     Serial.print(_DIST_MIN);
  Serial.print(",raw:");    Serial.print(dist_raw);
  Serial.print(",ema:");    Serial.print(dist_ema);
  Serial.print(",median:"); Serial.print(dist_median);
  Serial.print(",Max:");    Serial.print(_DIST_MAX);
  Serial.println("");

  // update last sampling time
  last_sampling_time += INTERVAL;
}

float find_median(float dist)
{
  if(cnt == N)
  {
    int del_id = upper_bound(arr_dist[id]) - 1;
    for(int i = del_id; i < N - 1; i++)
      arr_sorted[i] = arr_sorted[i+1];
    arr_sorted[N-1] = MAX;
  }
  arr_dist[id] = dist;
  int insert_id = upper_bound(dist);
  for(int i = cnt; i > insert_id; i--)
    arr_sorted[i] = arr_sorted[i-1];
  arr_sorted[insert_id] = dist;

  cnt = min(cnt + 1, N), id = (id + 1) % N;
  if(N % 2)
    return arr_sorted[cnt/2];
  else
    return (arr_sorted[cnt/2-1] + arr_sorted[cnt/2]) / 2;
}

int upper_bound(float value)
{
  int l = 0, r = cnt;
  while(l < r)
  {
    int m = (l + r) / 2;
    if(arr_sorted[m] <= value)
      l = m + 1;
    else
      r = m;
  }
  return r;
}

// get a distance reading from USS. return value is in millimeter.
float USS_measure(int TRIG, int ECHO)
{
  digitalWrite(TRIG, HIGH);
  delayMicroseconds(PULSE_DURATION);
  digitalWrite(TRIG, LOW);

  return pulseIn(ECHO, HIGH, TIMEOUT) * SCALE; // unit: mm

  // Pulse duration to distance conversion example (target distance = 17.3m)
  // - pulseIn(ECHO, HIGH, timeout) returns microseconds (음파의 왕복 시간)
  // - 편도 거리 = (pulseIn() / 1,000,000) * SND_VEL / 2 (미터 단위)
  //   mm 단위로 하려면 * 1,000이 필요 ==>  SCALE = 0.001 * 0.5 * SND_VEL
  //
  // - 예, pusseIn()이 100,000 이면 (= 0.1초, 왕복 거리 34.6m)
  //       = 100,000 micro*sec * 0.001 milli/micro * 0.5 * 346 meter/sec
  //       = 100,000 * 0.001 * 0.5 * 346
  //       = 17,300 mm  ==> 17.3m
}
```

```
컴파일 완료.

스케치는 프로그램 저장 공간 4980 바이트(15%)를 사용. 최대 32256 바이트.
전역 변수는 동적 메모리 504바이트(24%)를 사용, 1544바이트의 지역변수가 남음.  최대는 2048 바이트.
```

dist_ema는 필터링 된 값(dist_filtered)을 사용하지만,
dist_median은 측정한 값(dist_raw)을 그대로 사용한다.

// 값을 입력받고 최근 N개의 값의 중앙값을 반환하는 함수.
이미 N개의 값을 저장하고 있다면 N 번째 전 값을 arr_sorted[]에서 삭제한다.
이분탐색으로 삭제할 값의 위치 del_id를 찾고, del_id 이후의 값들을 한 칸씩 앞으로 당긴 후 맨 뒤에 MAX를 저장한다.

값을 입력받은 순서대로 arr_dist[]에 저장하고, 오름차순 정렬된 arr_sorted[]에서 dist보다 큰 첫 번째 값의 위치 insert_id를 찾은 후, insert_id부터 배열을 뒤로 한 칸씩 밀고 dist를 arr_sorted[insert_id]에 저장한다. 시간복잡도는 O(N)이다.

값의 개수 cnt와 arr_dist[]의 인덱스 id를 업데이트한다.
N이 짝수일 경우 중앙값 2개의 평균을 반환한다.

// 이분탐색을 실행하는 함수. arr_sorted[]에서, 입력된 값 value보다 큰 값 중 첫 번째 위치(인덱스)를 반환한다.
r의 초깃값이 cnt-1이 아니라 cnt인 이유는 cnt개의 값 중에서 value보다 큰 값이 없을 경우를 위해서이다.
시간복잡도는 O(logN)이다.