1. **Point Transformation(Process)**

**:** 주변 pixel value와 관련없이 pixel 하나 하나를 변형하는 기법.

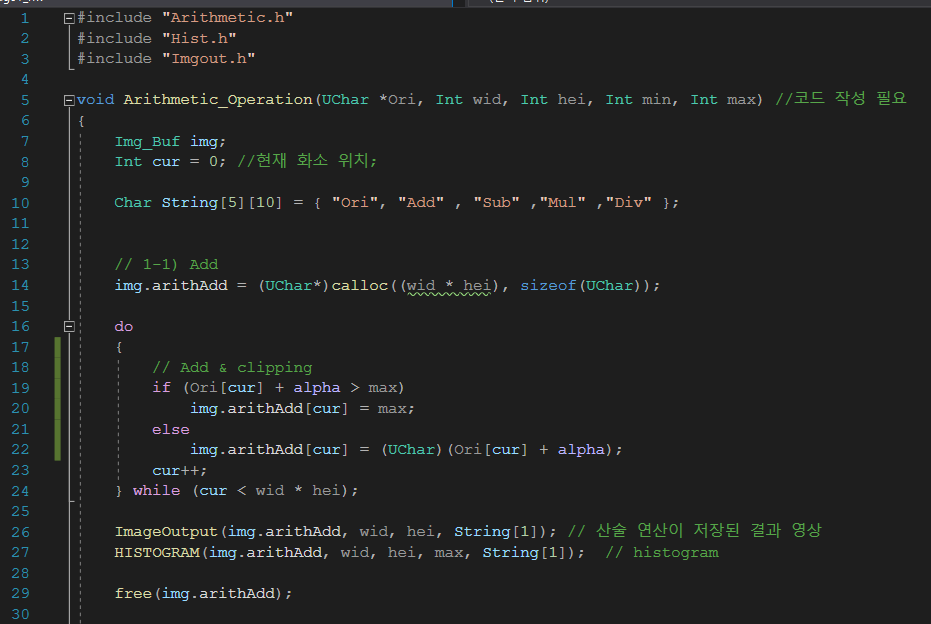
**실습 1) Arithmetic Operations(산술 연산)**

- 원본 화소 값에 **addition(+), subtraction(-), multiplication(\*), division(/)** 적용.

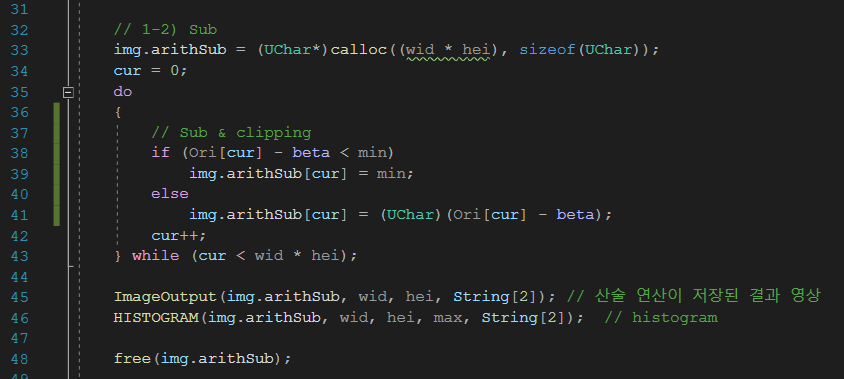
- 연산의 결과값이 최소값보다 작으면 최소값으로, 최대값보다 크면 최댓값으로 적용하는

**clipping** 과정과, 결과값에 소수점이 생길 때 **반올림**하는 과정이 필요합니다.

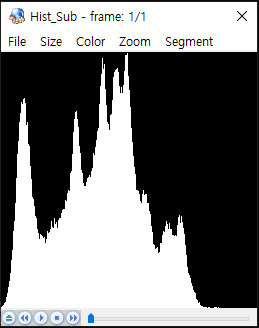
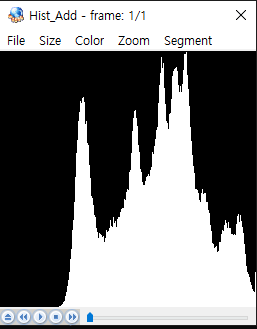
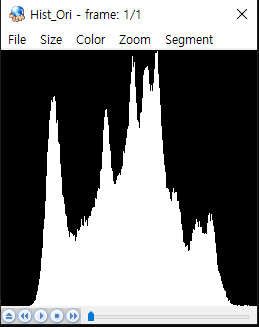
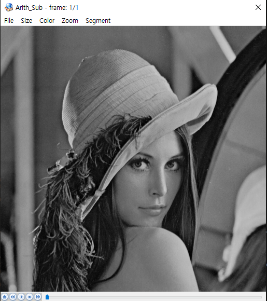
* **코드 분석과 결과 영상 및 히스토그램 비교**
* Addition(+) 과정입니다. Arithmetic.h 파일에서 alpha가 30이므로 addition 연산이 수행된 화소값이 화소의 최대값을 초과할 수 있기 때문에, 연산의 결과가 최대값을 초과하면 화소의 최대값으로 적용하는 clipping과정을 추가했습니다.



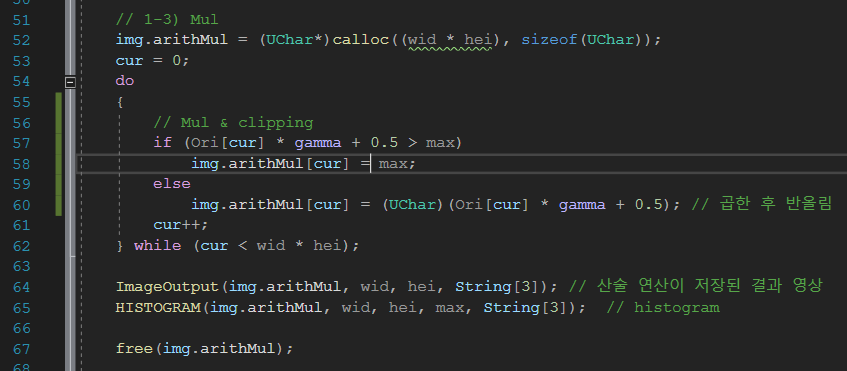
* Subtraction(-) 과정입니다. Arithmetic.h 파일에서 beta가 30이므로 Subtraction 연산이 수행된 화소값이 화소의 최소값보다 작을 수 있기 때문에, 연산의 결과가 최소값보다 작으면 화소의 최소값으로 적용하는 clipping과정을 추가했습니다.



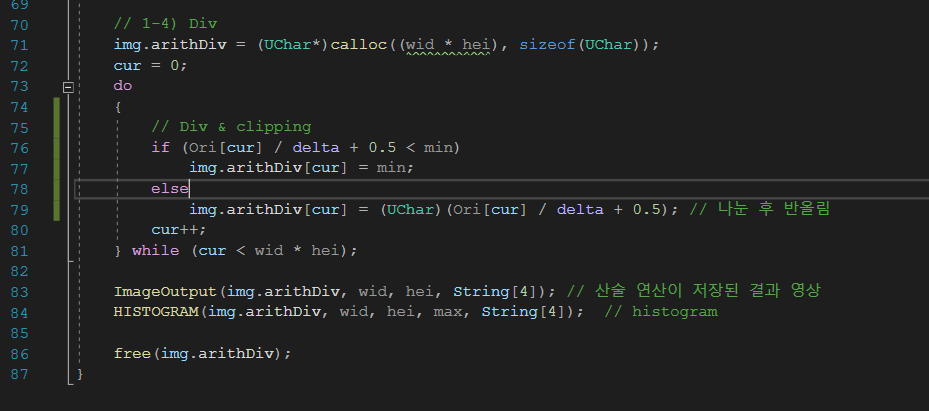
* 덧셈 연산을 수행했을 때 원본 영상보다 더 밝아지고, 뺄셈 연산을 수행했을 때 원본 영상보다 더 어두워집니다. 덧셈 연산의 히스토그램이 x축의 오른쪽으로 이동하며 밝은 화소 값을 원래보다 많이 갖게 되고, 뺄셈 연산의 히스토그램이 x축의 왼쪽으로 이동하며 어두운 화소 값을 원래보다 많이 갖게 된다는 사실을 확인 할 수 있었습니다. 즉, 덧셈과 뺄셈 연산은 영상의 밝기와 연관이 있습니다.



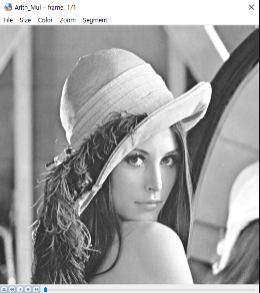
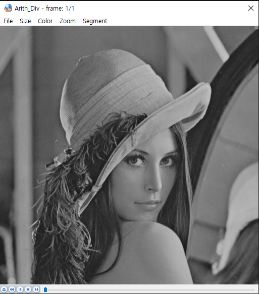
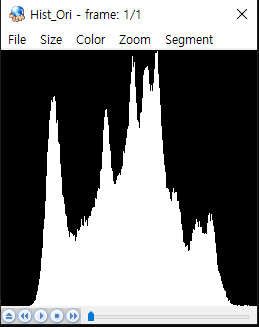
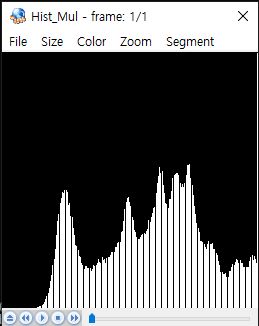
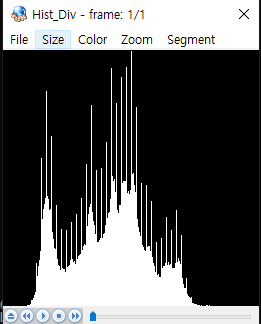
* Multiplication(\*) 과정입니다. Arithmetic.h 파일에서 gamma가 1.2이므로 multiplication 연산이 수행된 화소값이 화소의 최대값을 초과할 수 있기 때문에, 연산의 결과가 최대값을 초과하면 화소의 최대값으로 적용하는 clipping과정을 추가했습니다. 또한 연산의 결과가 소수점이 발생하는 경우가 있기 때문에 반올림 해주는 과정을 추가했습니다.



* Division(/) 과정입니다. Arithmetic.h 파일에서 delta가 1.2이므로 division 연산이 수행된 화소값이 화소의 최소값보다 작을 수 있기 때문에, 연산의 결과가 최소값보다 작으면 화소의 최소값으로 적용하는 clipping과정을 추가했습니다. 또한 연산의 결과가 소수점이 발생하는 경우가 있기 때문에 반올림 해주는 과정을 추가했습니다.



* 곱셈 연산을 수행했을 때 원본 영상보다 영상의 밝기 차이인 contrast가 증가하고 전체적으로 밝기가 증가하며 뚜렷해지는 효과가 있고, 나눗셈 연산을 수행했을 때 원본 영상보다 contrast가 감소하고 밝기가 감소하며 희미해지는 효과가 있습니다. Clipping 범위에 해당하는 화소가 많은 부분에는 화소의 최소값인 0과 최대값인 255의 양이 증가합니다. 따라서 곱셈과 나눗셈 연산의 히스토그램의 변화가 아래의 사진처럼 출력되었습니다. 곱셈의 경우 정규화를 진행했을 때 최대값인 255에 해당하는 화소들이 압도적으로 많기 때문에 상대적으로 다른 화소의 빈도수가 적어 그래프가 아래의 형태처럼 나타납니다. 곱셈, 나눗셈 연산은 영상의 선명도(contrast)와 연관있다는 사실을 확인할 수 있었습니다.

* Clipping 과정이 반복되므로, 추후에 clipping 과정을 함수로 적용시켜 영상처리를 수행하도록 코드를 작성해보겠습니다.

**실습 2) Histogram equalization**

- 불균등한 histogram을 균일하게 만들어주는 기법.

- 효과 : 영상의 pixel value를 균등하게 분산시킴으로써 낮은 contrast를 높여주어 명암 대비를 극대화

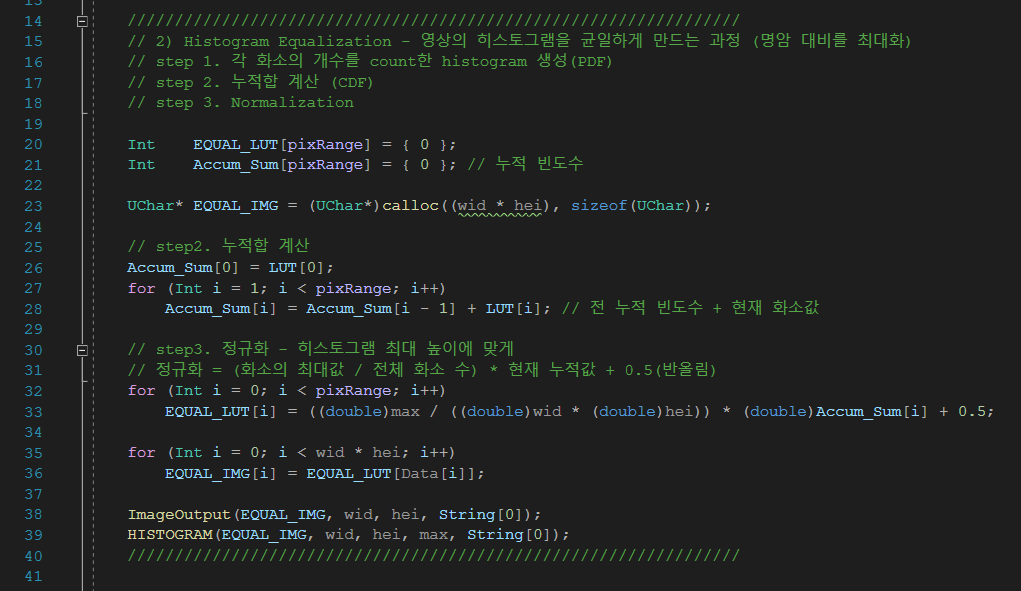
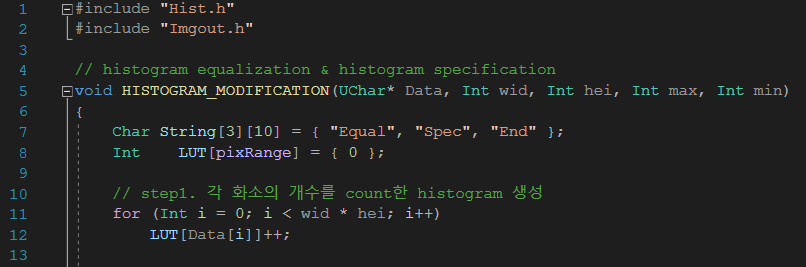
- 과정

1. 각 화소의 개수를 count한 histogram 생성(PDF)

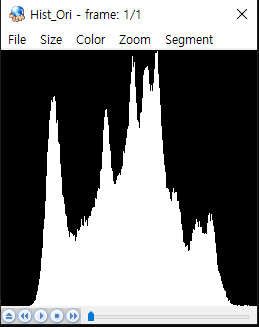
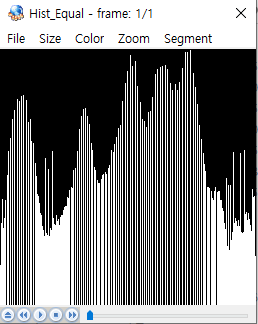
2. 누적합 계산(CDF)

3. LUT를 input영상에 적용 ( = 화소의 최대값/전체 화소 수 \* 현재 누적값(CDF) + 0.5(반올림) )

* **코드 분석과 결과 영상 및 히스토그램 비교**
  + 각 step에 해당하는 코드 내용은 가독성을 위해 주석으로 구분하였습니다.
  + LUT[]는 각 화소의 개수를 count한 히스토그램을, Accum\_Sum[]은 running sum을, EQUAL\_LUT[]는 평활화에 해당하는 lookup table을 뜻합니다.
  + 정규화는 화소의 최대값/ 전체 화소 수 \* 현재 누적값입니다. 이 과정에서 반올림을 위해 0.5를 더했습니다.



* 평활화를 적용한 영상의 히스토그램의 높이가 원본 영상의 히스토그램의 높이보다 상대적으로 일정합니다. 평활화 기법이 영상의 히스토그램이 균등하도록 pixel value를 분산시키는 기법이라는 것을 알 수 있었고, 원본 영상보다 평활화를 적용한 영상의 contrast가 높아져서 영상의 밝기 차이가 극대화 됨을 확인할 수 있었습니다.

**실습 3) Histogram specification** (=Histogram Matching)

- 원하는 히스토그램으로 변환하는 기법

- 효과 : 특정(원하는) 부분의 명암 대비를 극대화

- 과정

1. 원본 image의 histogram 생성(PDF)

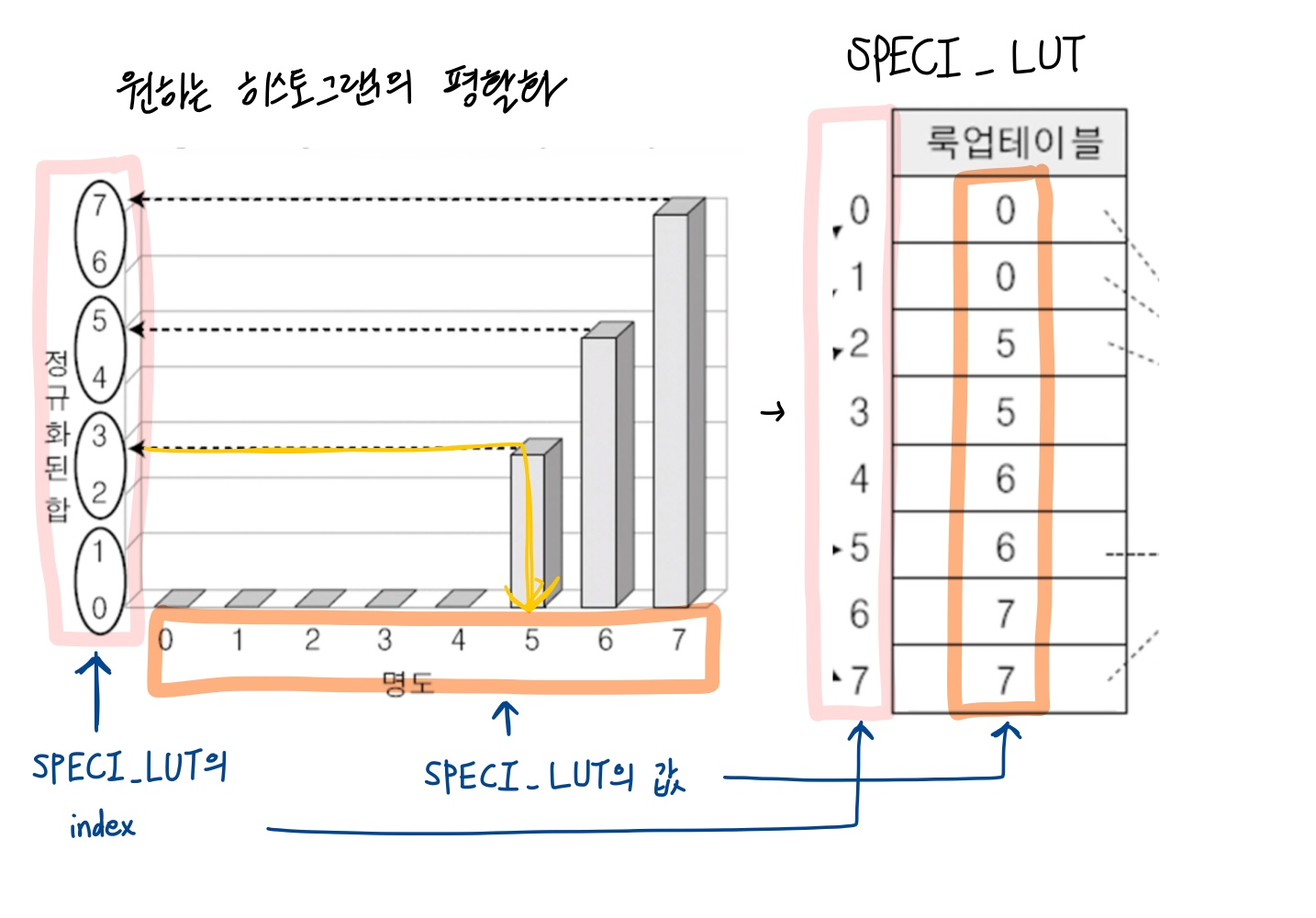
2. CDF & desired Histogram

3. desired histogram의 equalization

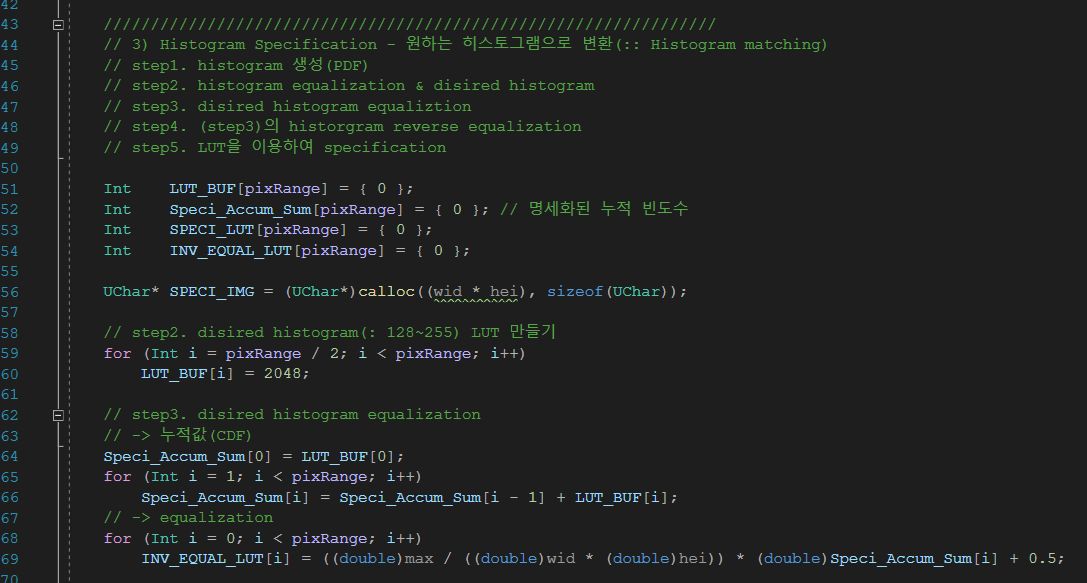
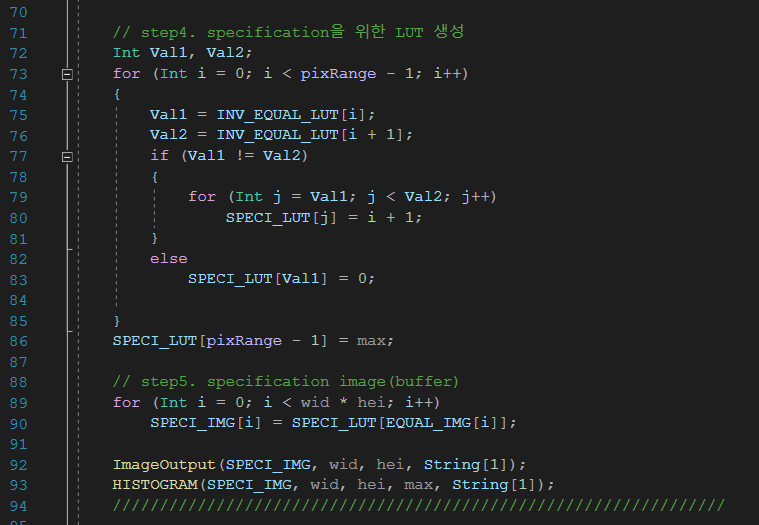
4. 3의 결과에서 reverse equalization

5. LUT 적용하여 원본 image의 PDF를 수정

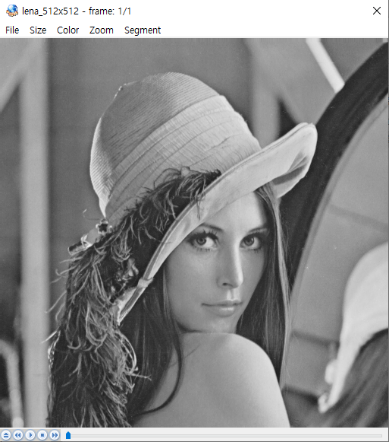
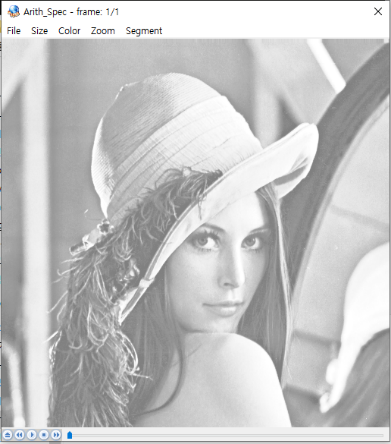
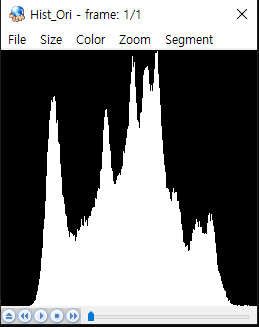
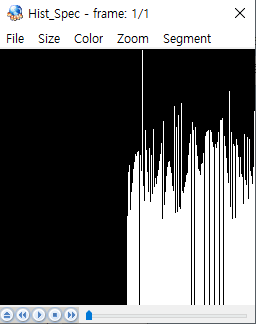
* **코드 분석과 결과 영상 및 히스토그램 비교**
  + 각 step에 해당하는 코드는 가독성을 위해 주석으로 구분 하였습니다.
  + 원본 영상의 히스토그램을 계산하고 평활화 하는 단계(step1-2)는 (실습 2)에서 진행했습니다.
  + Step2(Reference histogram) : LUT\_BUF[]는 얻고자 하는 형태(= desired/reference)의 PDF를 뜻합니다. 실습문제에서 reference histogram은 원래 pixel의 반의 범위(128~255)만 나타내는 히스토그램입니다.
  + Step3 : Reference histogram을 평활화 하기 위해 누적합을 구합니다. Reference histogram의 누적합은 Speci\_Accum\_Sum[] 변수에 저장되어 있고, 이를 평활화한 값은 INV\_EQUAL\_LUT[]에 저장되어 있습니다.
  + Step4 : specification을 위한 룩업테이블을 생성합니다. 3단계에서 생성한 reference histogram의 누적합이 LUT에서 index(=밝기값)이 되고, 원래의 밝기값(index)가 LUT에서 값(=reverse 평활화값)이 됩니다. Val1, Val2가 reference histogram의 누적합인데, 명세화 LUT(=SPECI\_LUT, j)의 index로 사용되고, 원래의 index(=i)가 SPECI\_LUT의 값으로 들어가 역평활화가 진행됨을 확인할 수 있습니다



* + Step5 : Step4에서 만든 specification 룩업테이블(SPECI\_LUT[])을 이용해 원본 영상의 평활화 값(EQUAL\_IMGP[)을 명세화하도록 바꿉니다.

**** 

* reference histogram과 명세화가 적용된 영상의 histogram의 범위가 128~255로 동일합니다. 또한 밝은 pixel value인 128~255 pixel 범위로 명세화가 적용되어 output 영상의 밝기가 원본 영상의 밝기보다 밝다는 것을 확인할 수 있습니다. histogram에서도 원본의 pixel범위보다 명세화가 적용된 histogram의 pixel범위가 밝은 쪽인 128-255 범위로 분포하기 때문에 명세화가 적용된 영상의 밝기가 좀 더 밝다는 것을 유추할 수 있습니다. 또한 원본 영상의 histogram이 명세화가 적용된 histogram보다 좀 더 다양한 pixel value를 사용하기 때문에 원본 영상이 명세화가 적용된 영상보다 더 contrast가 높습니다.

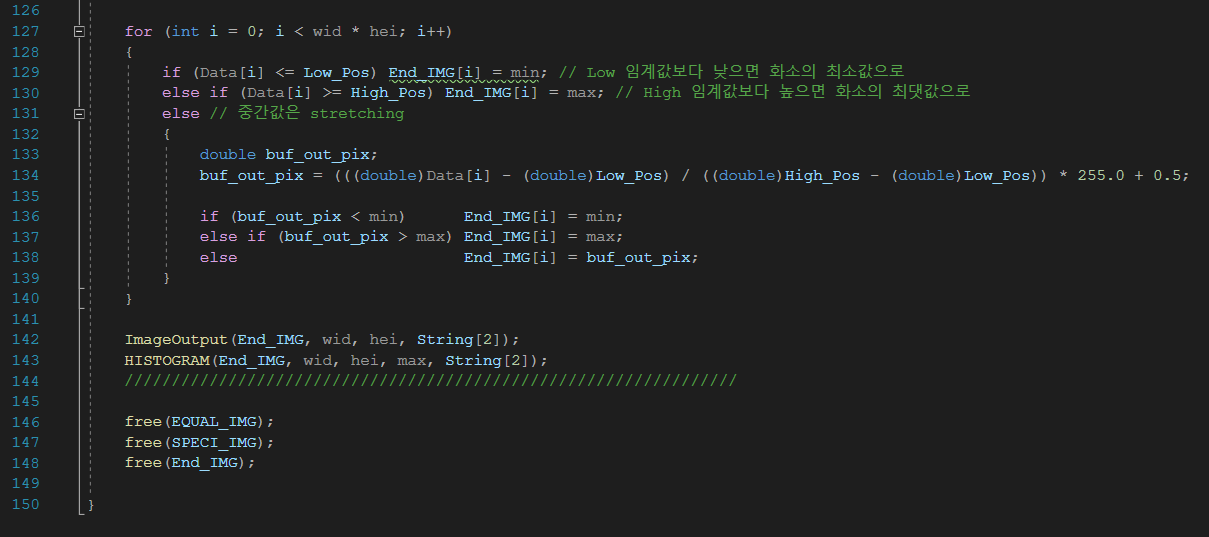
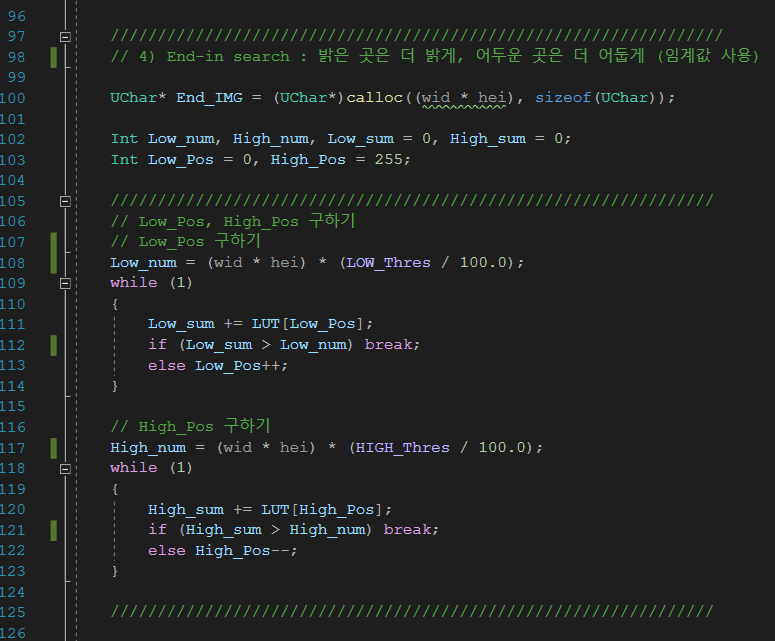
**실습 4) End-in search**

- 밝은 곳을 더 밝게, 어두운 곳을 더 어둡게 만드는 과정

- low 임계값(=30), high 임계값(=30) 사용하여 low 임계값보다 낮으면 화소의 최소값으로,

high 임계값보다 높으면 화소의 최대값으로 지정하고 그 중간값은 stretching합니다.

* **코드 분석과 결과 영상 및 히스토그램 비교**
  + Low 임계값보다 낮은 화소의 개수(=Low\_num)와 위치(=Low\_Pos), High 임계값보다 높은 화소의 개수(=High\_num)와 위치(=High\_Pos)를 구해야 합니다. 각 임계값에 해당하는 화소의 개수는 전체화소 \* 임계값/100로 구할 수 있습니다. Low 임계값에 해당하는 화소의 위치를 알기 위해 화소의 밝기 0부터 시작하여 해당하는 개수만큼 누적하고, 그 누적된 값이 임계값에 해당하는 화소의 개수보다 많아질 때까지 pos를 하나씩 증가하며 반복합니다. High 임계값에 해당하는 화소의 위치를 알기 위한 방법도 높은 화소 밝기의 위치를 알기 위한 과정이기 때문에, pos를 255부터 시작하고 반복문이 진행될 때 pos를 하나씩 감소시킨다는 점만 다르고 전반적인 과정은 동일합니다.
  + 이렇게 구한 임계값의 위치를 참고하여, low 임계값의 위치보다 낮으면 화소의 최소값으로 지정하고, high 임계값의 위치보다 높으면 화소의 최대값으로 지정합니다. 그 사이 값은 (pixel data – low\_Pos)/(High\_Pos – Low\_Pos) \* max + 0.5(반올림)으로 지정합니다.



* Low 임계값보다 낮으면 흰색으로 지정되고, high 임계값보다 높으면 검정색으로 지정하기 때문에 결과 histogram의 분포가 좀더 균일합니다. 중앙으로 몰려 있는 histogram을 균일하게 늘려주었기 때문에 더욱 다양한 pixel value 범위를 사용하게 되고, contrast가 더욱 높아집니다. 원본 영상과 end-in search를 수행한 영상을 비교해보면 원본 영상보다 더욱 밝기 대비가 뚜렷함을 확인할 수 있습니다. 