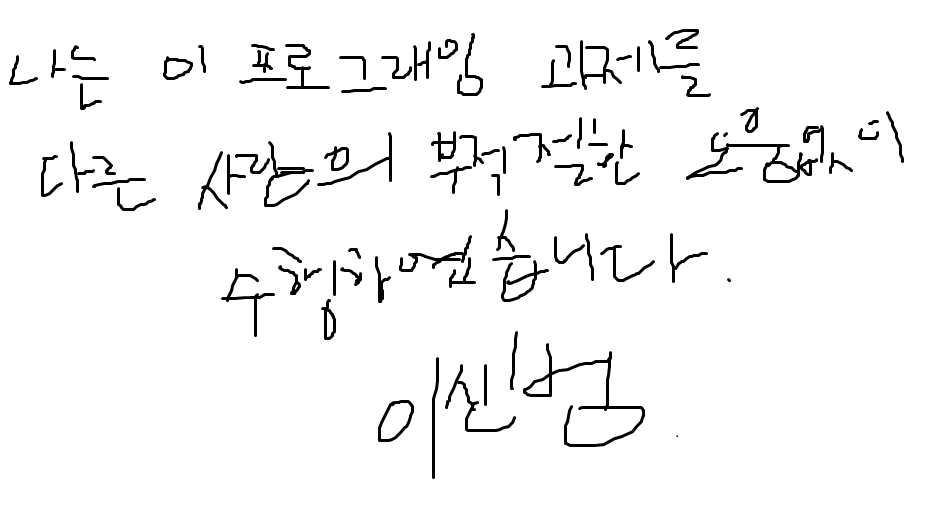
**ASSIGNMENT #4**

지도교수 : 윤은영 교수님

학생 이름 / 학번 / 학과 : 이신범 / 20180285 / 무은재 학부

POVIS ID : nm160

Problem #2

1. 개요

어싸인 4는 ‘파일 입출력’, ‘동적할당 및 free’, ‘구조체 사용’을 정확히 이해하고, 3차원 채널로 구성된 CUBIC 을 활용하여 사진을 변환하는 프로그램을 제작하는 것이었다.

CUBIC을 2차원 배열로 관리하고, CUBIC 내부에 있는 data를 또 3차원 배열로 관리했다.

사실 data의 경우 3차원 배열을 활용하지 않고 1차원 배열 여러 개를 x,y,z 축과 같이 활용할 수 있었을텐데, 다차원 동적할당을 활용해보라는 의미로 float\*\*\* 을 주신 것 같다 (어려웠다)

Data의 변환 같은 경우 ‘컨볼루션’ 이라는 행렬 계산을 이용했다.

1. 알고리즘

<pseudocode>

void copy\_for\_remove(int num\_cubic, int select\_image, CUBIC\*\*\* put); // 바뀌기 전 int num\_cubic을 받는다.

// 임의의 CUBIC[4]를 지우면 [5] [6] [7]... 을 앞으로 땡겨준다.

void realloc\_func(int num\_cubic, CUBIC\*\*\* put); // main 에서 num\_cubic ++ 혹은 -- 이후 호출

// 전달받은 int num\_cubic을 인식해서 Realloc을 진행한다.

CUBIC\* image\_load(FILE\* input, char\* name\_put); // 메인에서 realloc\_func 이후 호출

// FILE\* input 으로 파일을 읽고, CUBIC의 멤버를 채워서 CUBIC\*을 반환한다.

void image\_save(CUBIC\* selsected\_image, char\* name\_put); // main 에서 scnaf(&select), scanf(&name) 받는다.

// FILE\* output으로 CUBIC\* image[select] 내용을 "name" 이라는 파일명으로 저장한다.

CUBIC\* filter\_load(FILE\* input, char\* name\_put); // realloc\_func 이후 메인에서 호출한다.

// 위의 image\_load와 동일하다.

void print\_cubic(int num\_cubic, CUBIC\*\* put); // CUBIC의 목록을 출력한다.

// 인자로 주어지는 CUBIC\*\* put 에 따라서 image 출력과 filter의 출력을 모두 할 수 있다

void normalize(CUBIC\* image); // main 함수에서 print\_func 진행하고 normalize 할 인수를 scanf(&select) 한다.

// CUBIC[select] -> data 를 normalize 한다.

void denormalize(CUBIC\* image); // normalize 함수와 동일하다.

// CUBIC[select] -> data 를 denormalize 한다.

CUBIC\* image\_convolution(CUBIC\* selected\_image, CUBIC\* selected\_filter);

// 메인함수에서 scanf(&select1), scanf(&select2) 하고, realloc\_func 이후 호출

// 선택된 두 CUBIC[select]->data 를 통해 convolution을 진행하고 CUBIC\* 반환

void free\_data(CUBIC\* selected); // CUBIC\* 내부의 data 및 CUBIC\* 동적할당 해제

이번 어싸인에서는 다음의 함수를 이용했다. Main 함수를 작성하기 전에 함수 프로토타입을 모두 완성시켜놓은 뒤, 코딩을 진행하니 전체적인 구조를 잡기에 수월했다.

Main()

{

Printf 메뉴 출력

Scanf 메뉴 입력받기

Switch(메뉴입력) 으로 1 ~ 10 까지의 동작을 수행한다.

Case 1:

FILE \* input

Scanf(파일명)

파일명이 존재하는지 체크

Num\_image++ ( 여기까지 왔다는 것은, 파일이 존재해서 load 할 것이라는 거니까 )

Realloc\_func ( num\_image가 변화했으니, realloc 을 해주어야 한다. )

While(!(temp[i] == ‘.’))

Name\_put[i] = temp[i] // 파일명 ( aaa.ppm ) -> “aaa” 만 저장한다.

I++

Images[num\_image -1] = image\_load ( CUBIC\* 을 반환받는다. )

Fclose(input)

Case 2:

Print\_cubic // CUBIC 목록 출력

Scanf(인덱스 입력받기)

오류체크 // 잘못된 인덱스 입력했을 경우

Scanf(저장할 파일명 입력받기)

Image\_save

Case 3:

Scanf(삭제할 인덱스 입력받기)

오류체크 // 잘못된 인덱스 입력했을 경우

Free\_data // CUBIC 내부의 data 동적할당 free 해주기

Free(CUBIC) // CUBIC 자체의 동적할당 free 해주기

Copy\_for\_remove // 삭제된 인덱스 이후의 CUBIC 들을 한칸씩 땡겨준다.

Num\_image –

Realloc\_func // num\_image – 되었으니, realloc 해주기

Case 4:

Case 1: 과 동일하다. ( images 가 아닌 filters 를 다룬다는 점을 제외하면 )

Case 5:

Case 3: 과 동일하다. ( images 가 아닌 filters 를 다룬다는 점을 제외하면 )

Case 6:

Printf ========IMAGE LIST==========

Print\_cubic

Printf ========FILTER LIST===========

Print\_cubic

Case 7:

Print\_cubic

Scanf(정규화 할 이미지 인덱스 입력받기)

에러체크 // 잘못된 인덱스 입력했을 경우

에러체크 // 이미 정규화된 인덱스를 정규화하려고 했을 경우

Normalize // 정규화 시킨다

Case 8:

Print\_cubic

Scanf(역정규화 할 이미지 인덱스 입력받기)

에러체크 // 잘못된 인덱스 입력했을 경우

에러체크 // 이미 역정규화된 인덱스를 역정규화하려고 했을 경우

Normalize // 역정규화 시킨다

Case 9:

Print\_cubic

Scanf ( 컨볼루션 할 이미지 인덱스 입력받기 )

에러체크

Print\_cubic

Scanf( 컨볼루션 할 필터 인덱스 입력받기 )

에러체크

Num\_image ++ // 컨볼루션 할 경우에도 인덱스가 하나 추가된다

Realloc\_func

Images[num\_image -1] = image\_convolution // 컨볼루션 진행

Is\_normalize == 1 일 경우 파일 명 뒤에 \_blur 추가

Is\_normalize == 0 일 경우 파일 명 뒤에 \_edge 추가

Case 10:

Images 내부의 data 동적할당 free

Images 내부의 CUBIC 동적할당 free

Images 동적할당 free

Filters 내부의 data 동적할당 free

Filters 내부의 CUBIC 동적할당 free

Filters 동적할당 free

Return 0;

Default: (목록 1~10 이외의 숫자를 입력했을 경우)

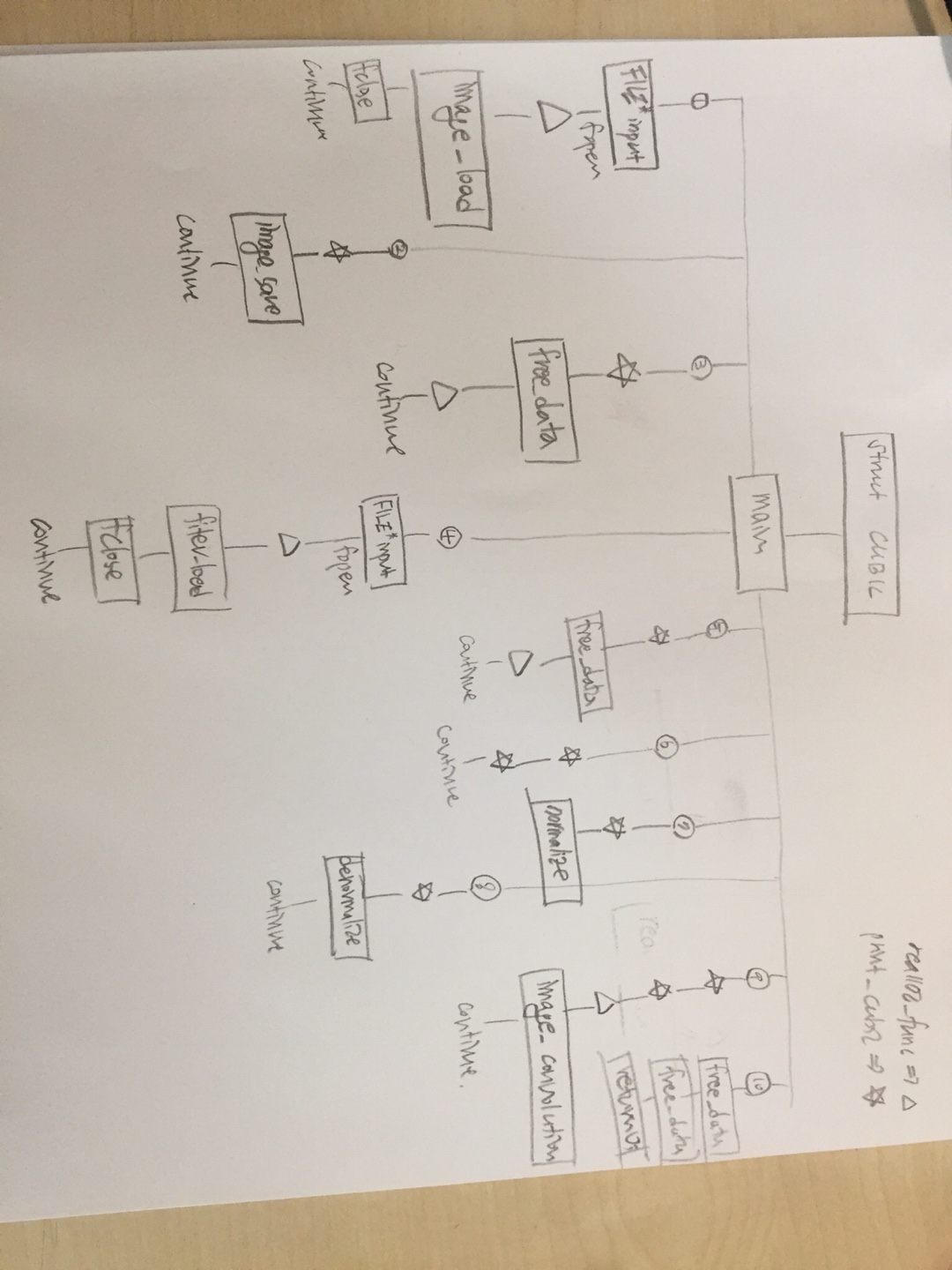
Printf(올바른 값을 입력해주세요.)

Continue;

}

1. 전체구조

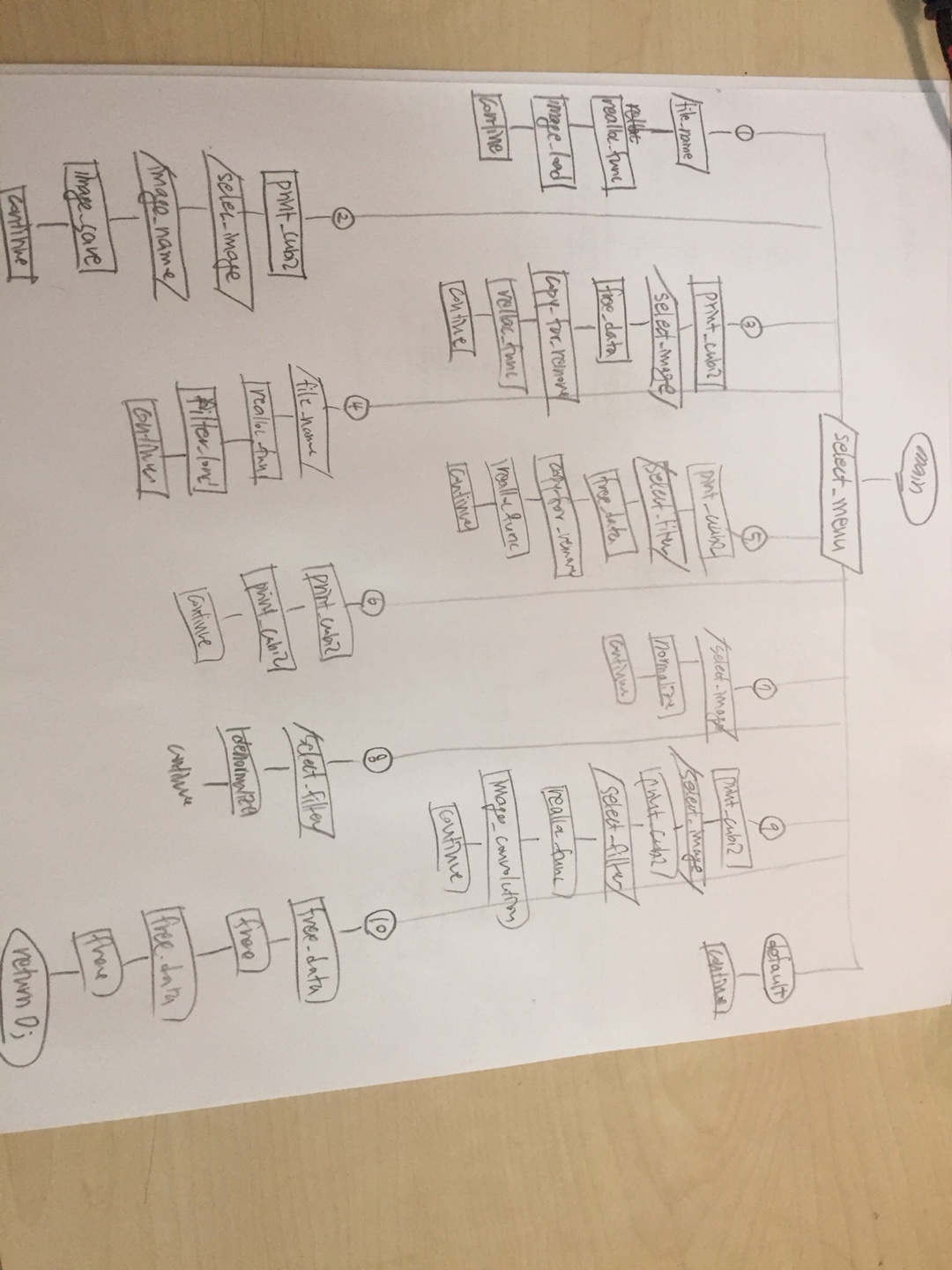
<structure chart>



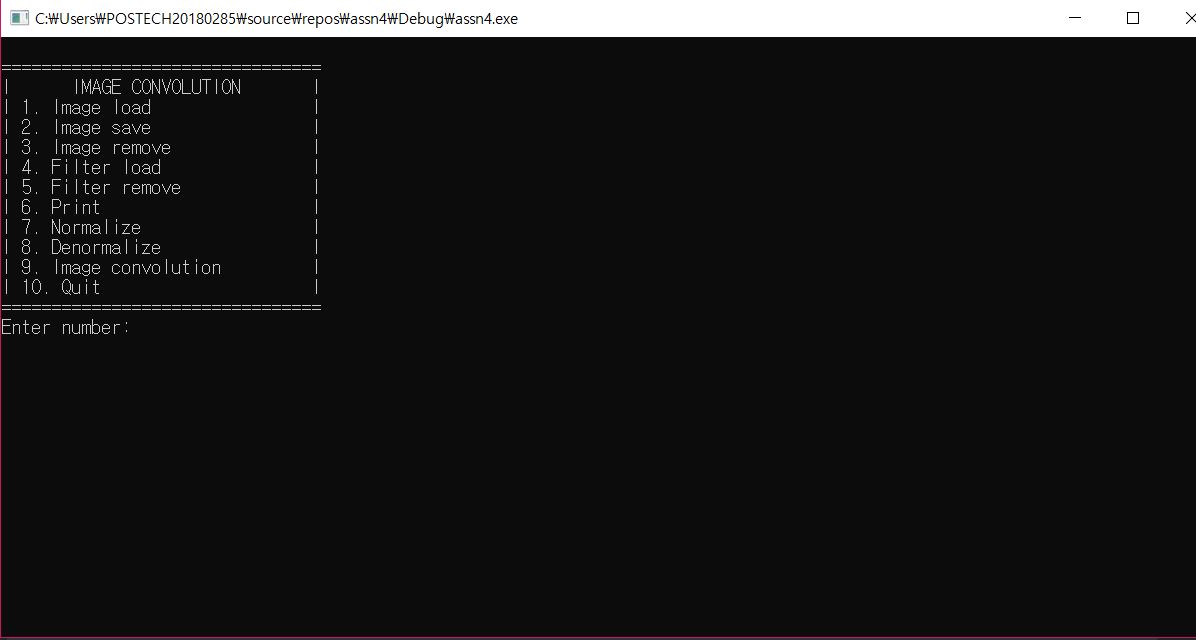
Print\_CUBIC 함수와

Realloc\_func 을 제외하고 대부분의 함수가 독립적으로 작용하고 있음을 알 수 있다.

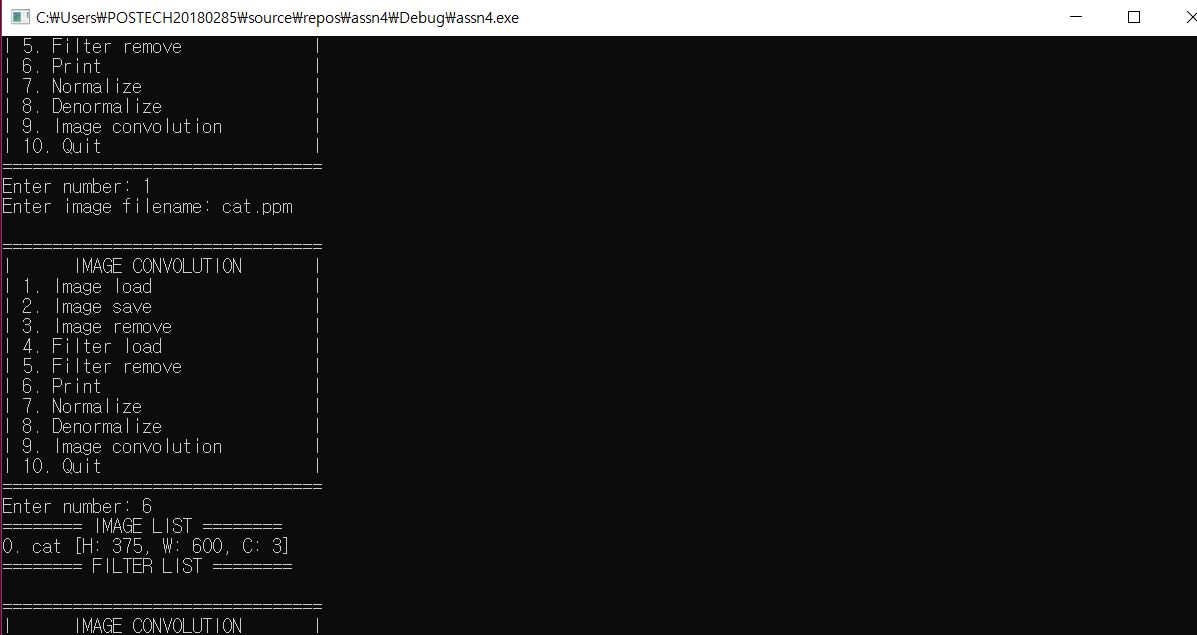
<flow chart>



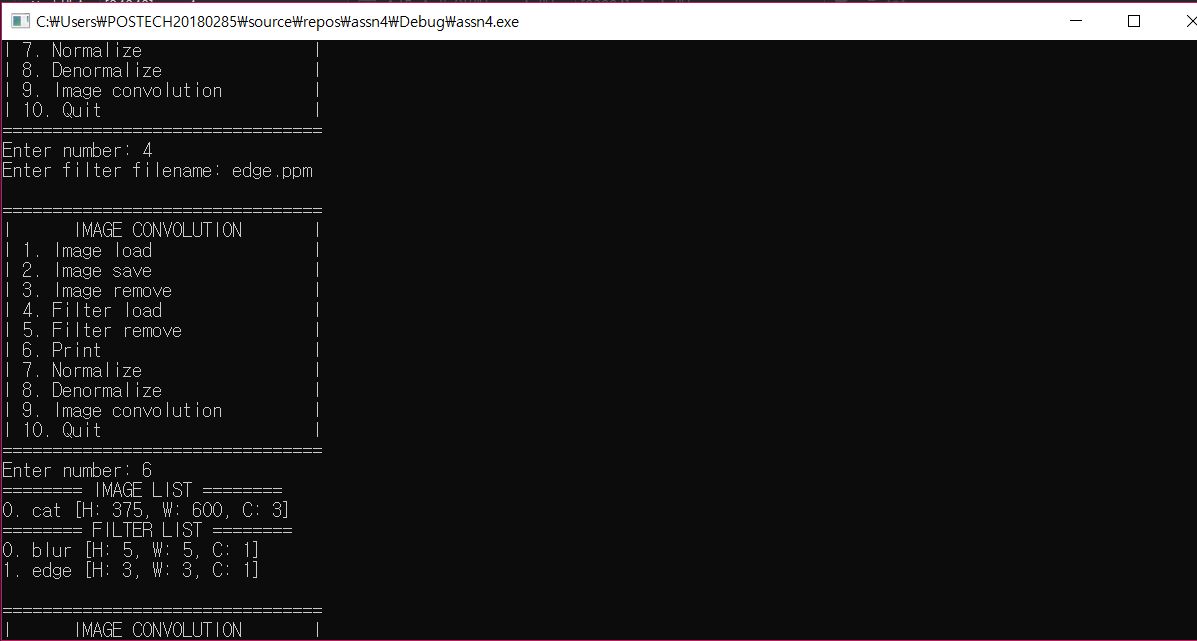
1. 실행예제



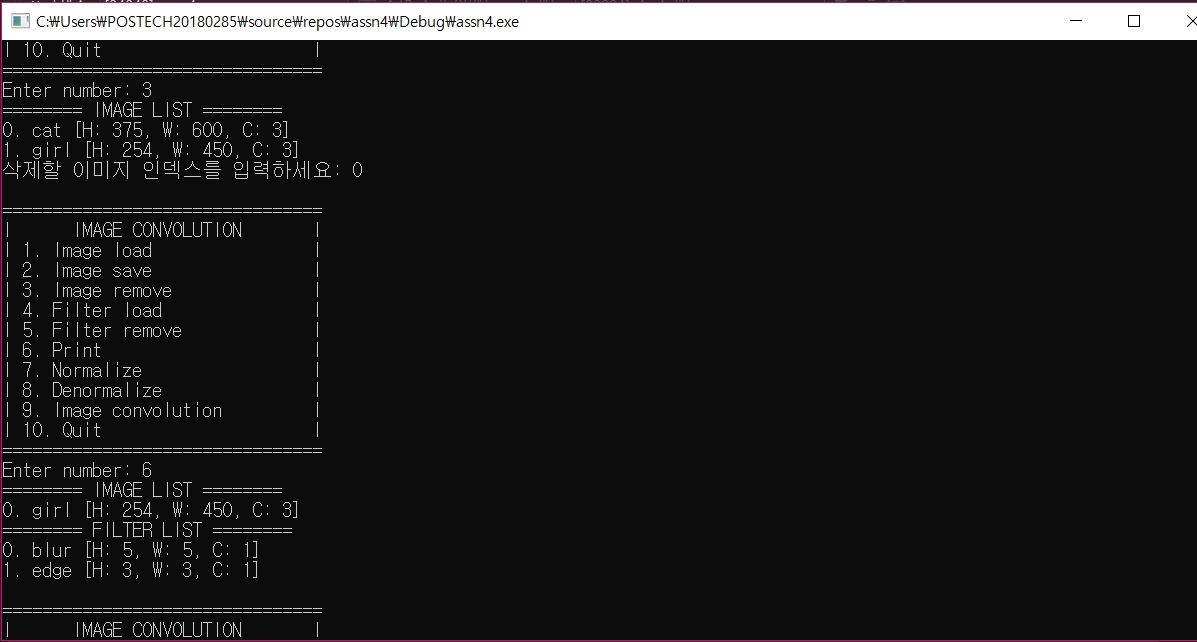
프로그램을 켜보았다.



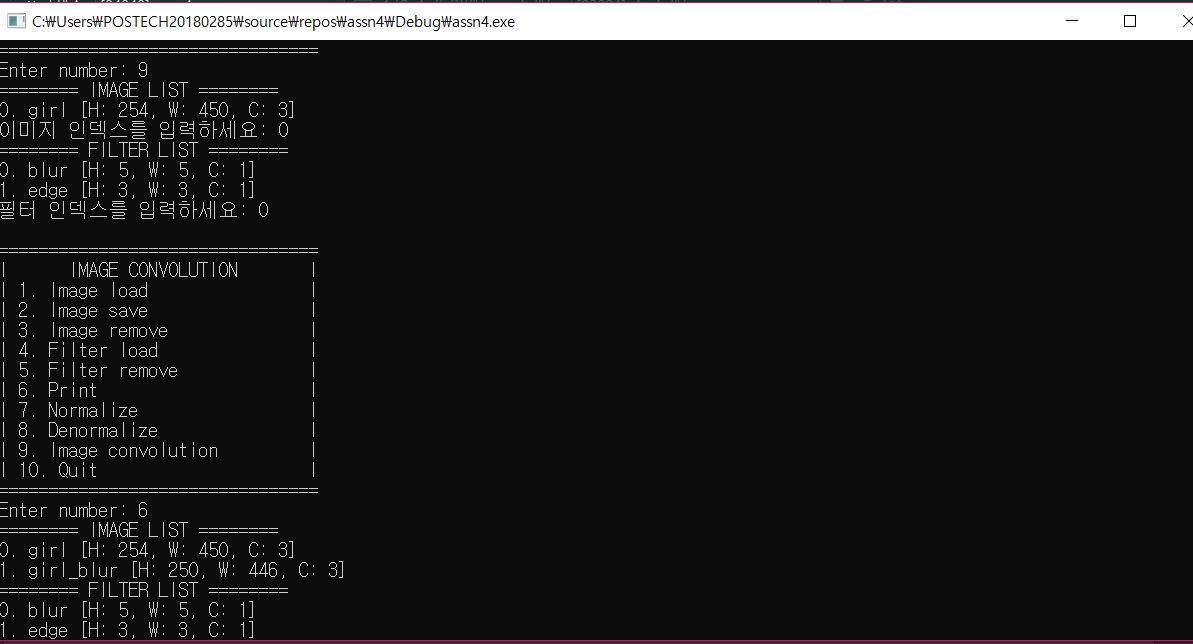
1을 입력해서 이미지 로드를 했다. Print 를 해보니, cat.ppm이 잘 저장되었다.



4를 입력해서 필터 로드도 해보았다. 역시 잘되었다.



Girl.ppm을 로드한 뒤, 3을 입력해서 cat.ppm 을 지워보았다. Girl의 인덱스가 1에서 0으로 간 것을 보니, 잘 지워졌다.

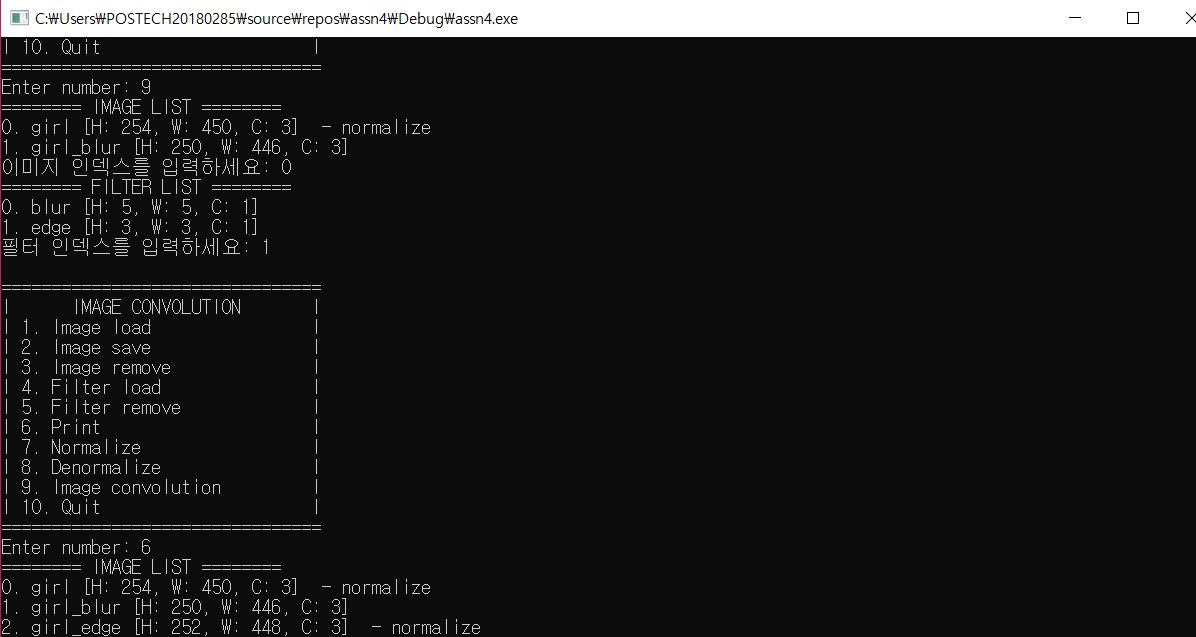


9를 눌러서 정규화되지 않은 girl.ppm 을 컨볼루션 해보았다.

Girl\_blur 이 잘 만들어졌다.

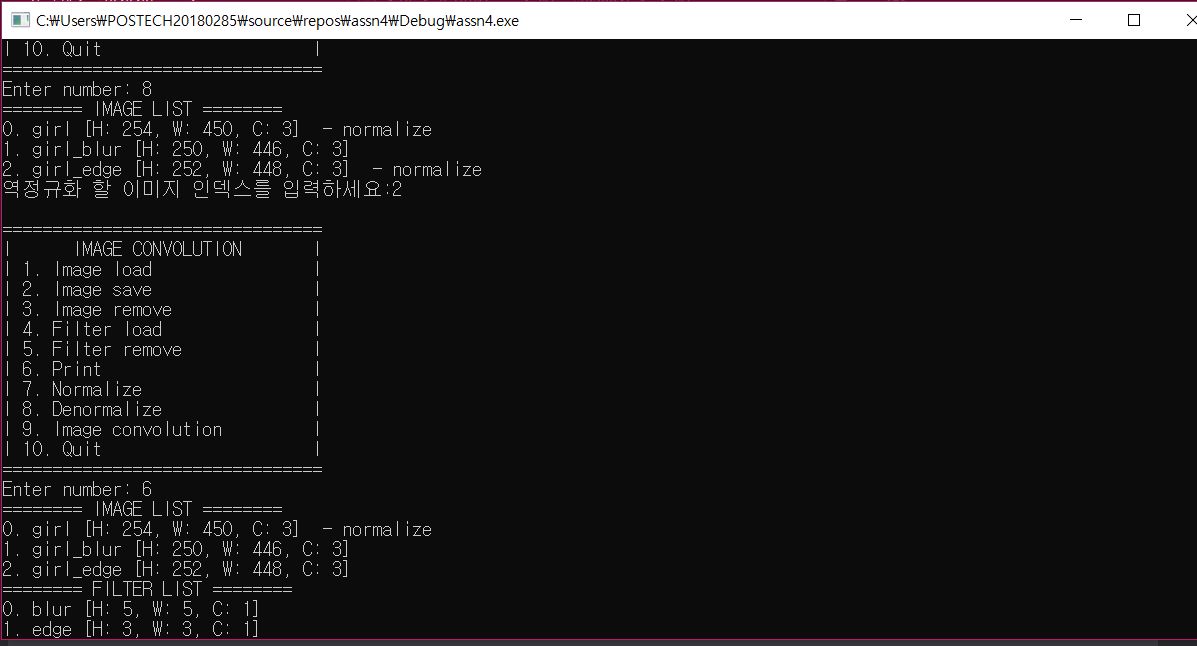


7을 입력해서 girl 을 정규화 해보았다. 잘 정규화 되었는지, - normalize가 써졌다.



정규화된 girl.ppm 을 edge 필터를 이용해 컨볼루션 해보았다.

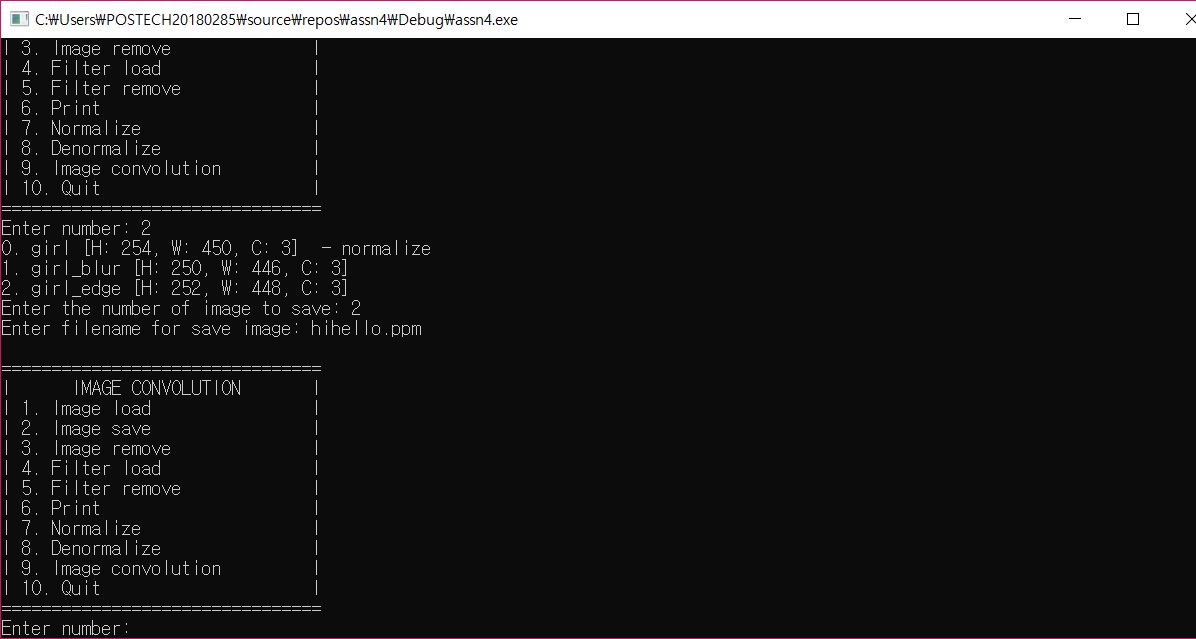
Girl\_edge 가 생겼다 ( 아직 denormalize 안했다는 것을 알 수 있다. )



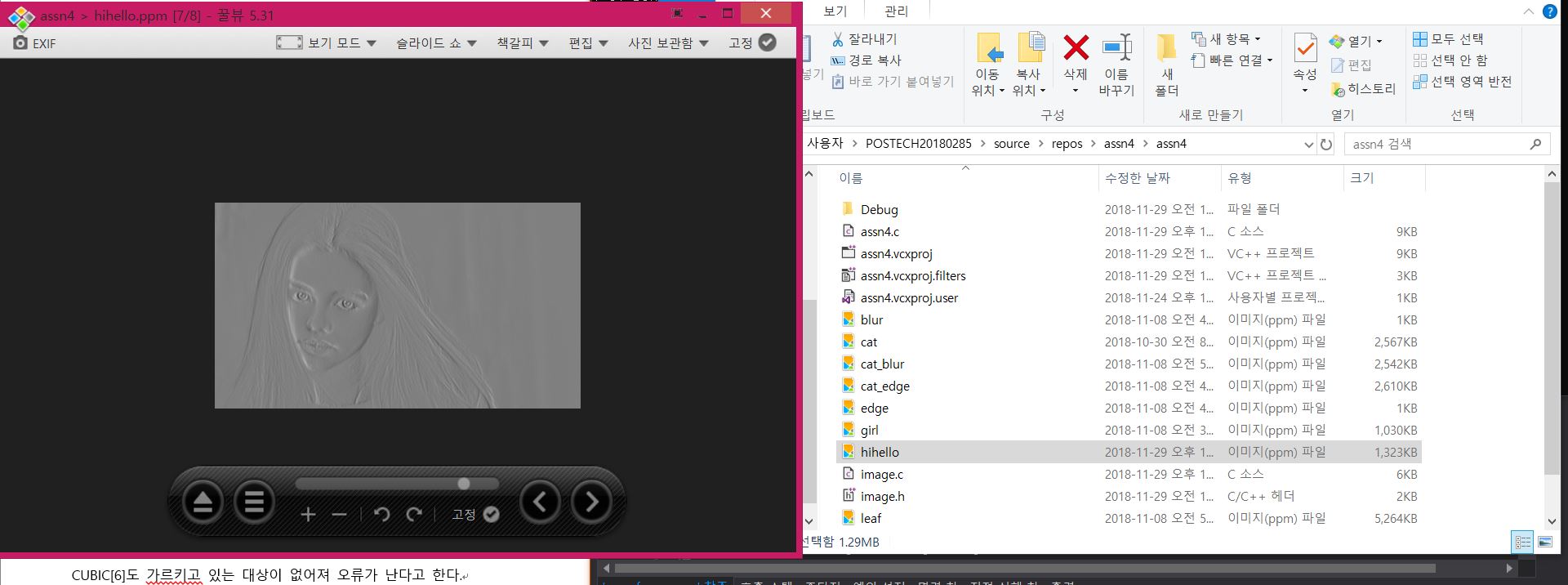
8을 입력해서 girl\_edge 를 denormalize 해주었다. – normalize가 사라졌으니, 잘 된 것 같다.



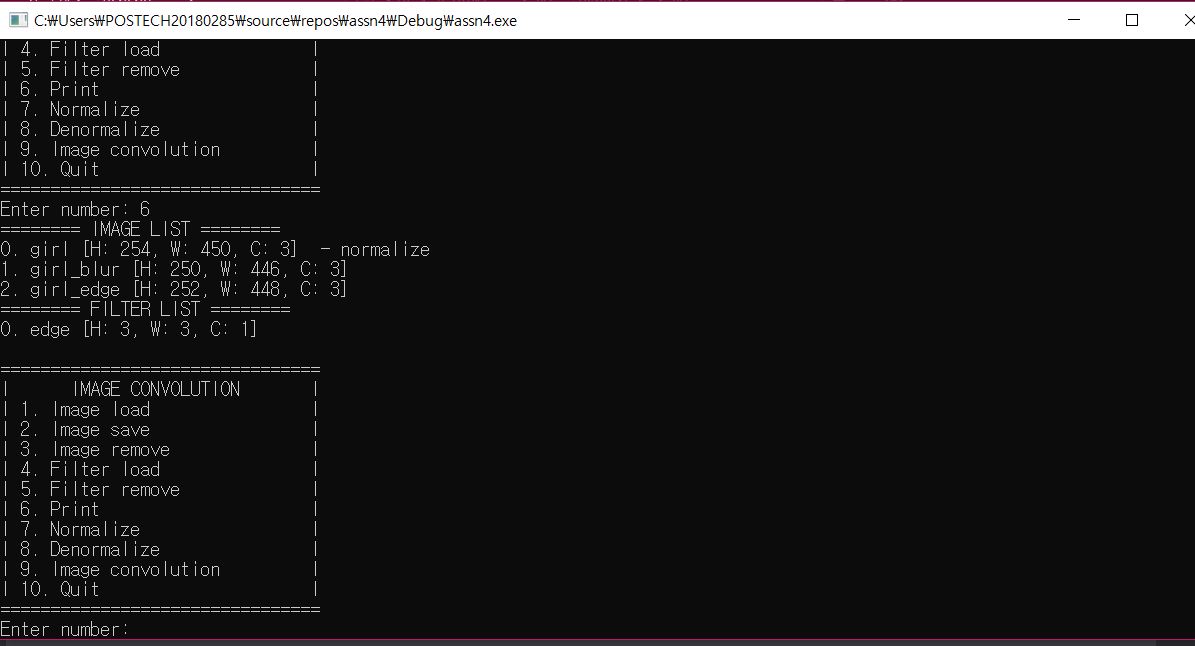
6을 입력해서 한 번 프린트 해보았다. 잘 나온다.



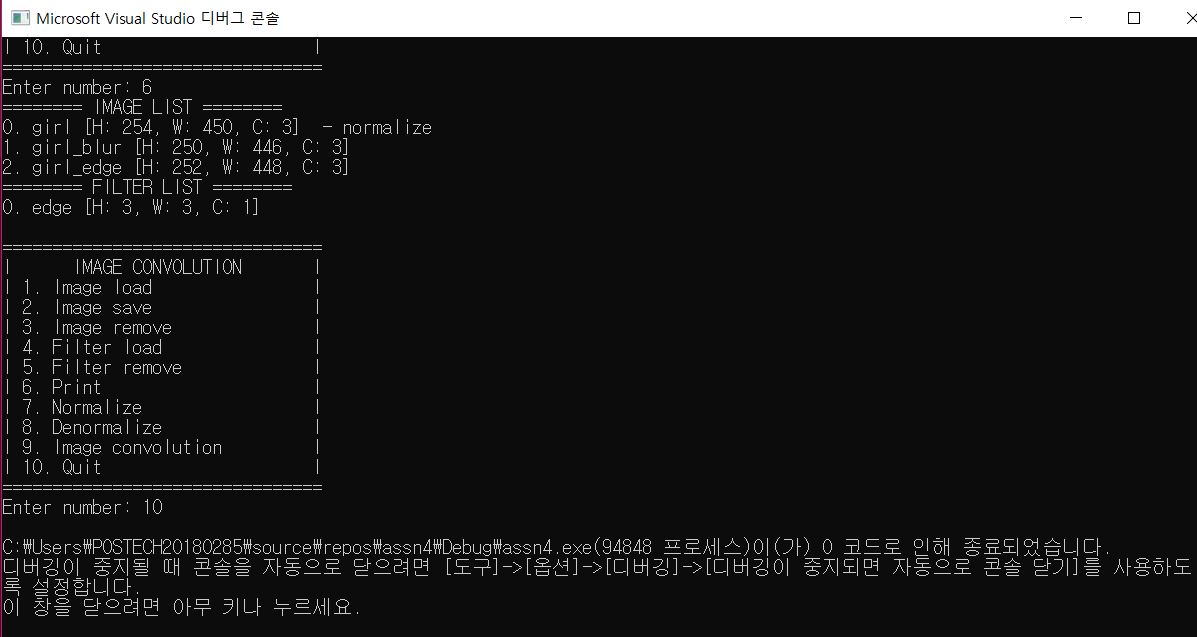
2를 입력해서 girl\_edge 를 hihello.ppm 으로 저장해보려고 한다.



음, 잘 저장되었다!

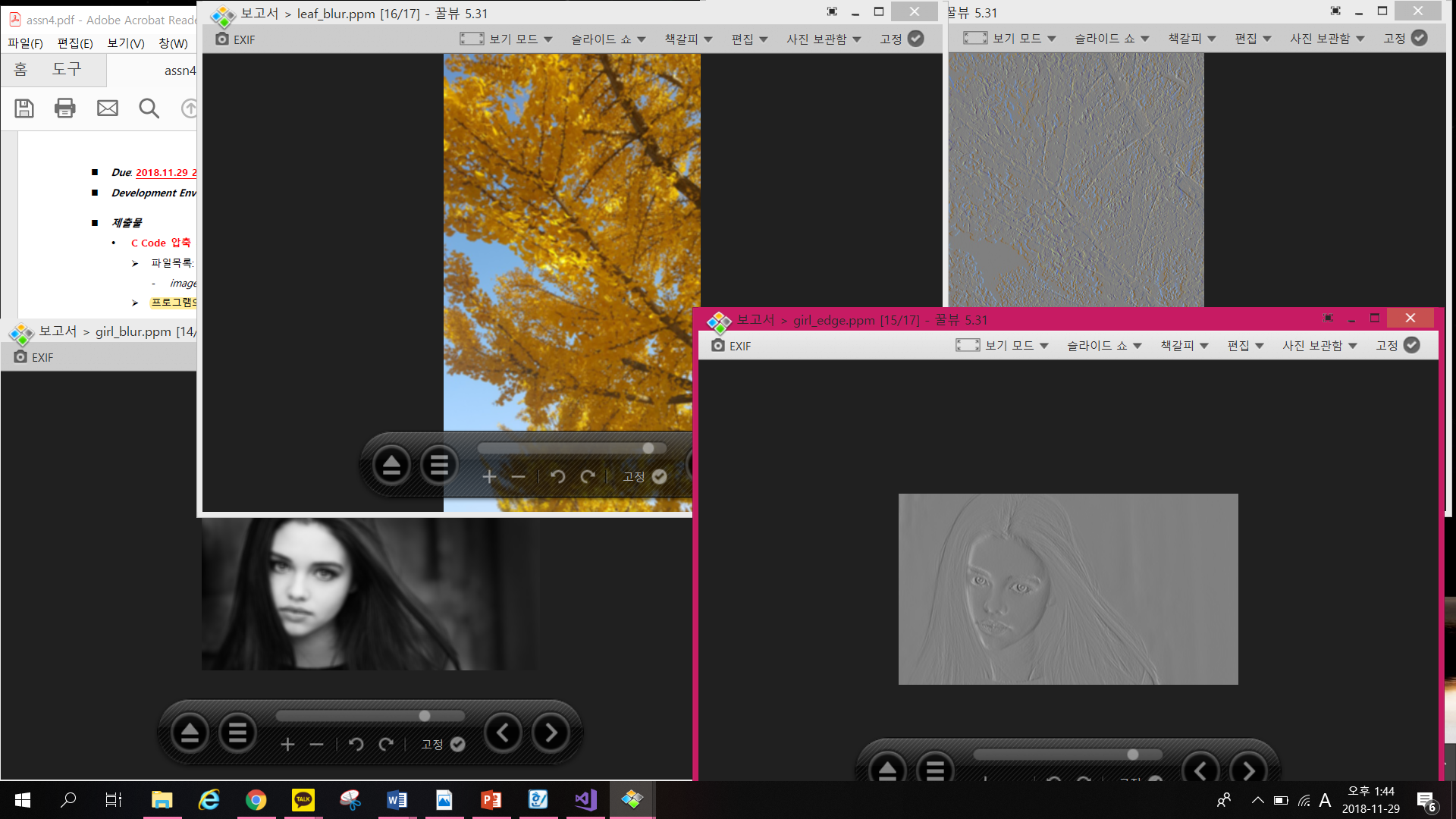


6을 입력해서 필터도 한 번 지워보았다. 잘 지워진다.



모든 기능이 잘 돌아간다.

주어진 에러 상황에 대해서도 올바른 에러 메시지가 출력되도록 코딩했다!



잘 blur 와 edge가 저장되었다.

1. 토론

이번 어싸인 기간에 내가 속해있는 학생회의 재정 심의가 겹쳐서 어싸인을 미리 해놓을 시간이 없었다. 11.27 부터 어싸인 pdf를 보고 프로토 타입 작성을 시작했는데, 주변에서 이번 어싸인이 엄청나게 어렵다는 말을 많이 해서 너무 걱정이 되었다.

다른 어싸인들에 비해서 엄청 어려웠던 것은 사실이나, 2~3 차원 동적할당이나, free를 관리하는 것 등 기초적인 개념을 이해하고 나니, 함수를 작성하고, main 함수와 각각 함수간의 관계를 생각하는 것은 그리 어렵지 않았다.

일단 어싸인을 하면서 어려웠던 점, 헷갈렸던 점은 다음과 같았다.

1. Free와 copy\_for\_remove 를 해주는 과정에서 오류가 났다.

원래는 select\_image가 있으면, select\_image 를 free 하지 않고, select\_image 이후의 함수들을 copy\_for\_remove 를 통해 앞에 것을 덮어씌우고, 맨 뒤에 있는 CUBIC\* 을 free 해주었는데, 할당 오류가 났다.

원인을 찾아보니, 만약 CUBIC[7] 까지 있다면, 내 방식대로 copy\_for\_remove를 해주게 되었을 때는 CUBIC[6], CUBIC[7]이 얕은 복사가 되어있는 상태라, CUBIC[7]을 free 해주게 되면 CUBIC[6]도 가르키고 있는 대상이 없어져 오류가 난다고 한다.

그래서, 방식을 바꾸었는데, 이 것을 발견하는 것이 매우 어려웠다.

1. 연산자 우선 순위가 어려웠다.

예를들어 put[i]->name 이라고 해놓고 실행하면, 에러가 걸렸는데, 나중에 원인을 찾아보니 (put[i])->name 이라고 해야했다. 이와 마찬가지로, copy\_for\_remove 함수에서도

\*put[i] 를 (\*put)[i] 라고 써주니까 실행이 되었던 것 같다!

1. 동적할당의 원리를 이해하고, 이를 free 시키는 것이 어려웠다.

지난 LAB 과제였던 2차원 동적할당도 어려웠는데, 3차원 동적할당을 하라니,, 이것과 call by address가 겹쳐서 \*을 몇 개 써야하지? 라는 고민을 너무 많이 했다. 헷갈리는 것도 많고, 어려웠다.

Free와 같은 경우, 맞게 free를 했는지 체크하기 위해 전역변수 int count = 0; 을 지정하고, malloc 과 free를 할 때 바로 밑에다가 count ++; , count --; 를 해서 올바르게 free가 되었는지 체크했다.

Realloc 을 진행하는 경우를 제외하고 모두 count 가 올바르게 나와서, free 를 맞게 했구나, 하고 알 수 있었다.

1. 인덱스 이름을 정할 때, cat.ppm 이라고 받은 것 중에서 cat 만 골라서 넣는 것이 어려웠다.

의외로 단순할 것 같았는데, 생각하는데 시간이 좀 걸렸다.

While 문을 돌려서 ‘.’ 이 나올 때 까지, name\_put[i] = temp[i] 를 해주었더니,

Cat.ppm 중 cat 만 name\_put에 저장할 수 있었다!

그리고 잊지 않고, name\_put 마지막에 NULL 을 넣어주었다.

1. 결론

동적 할당은 수많은 데이터를 관리할 때 저장공간을 아낄 수 있는 유용한 기법이지만, 그것을 다루기 위해서는 정확한 이해가 필요하다는 것을 알았다. 우리가 원래 하던 방식대로 엄청나게 큰 배열을 잡아놓고 진행하면, 작성하기는 쉽지만 컴퓨터가 그것을 실행하는 과정에서 많은 데이터 낭비가 이뤄질 것이다.

동적할당은 프로그램의 크기가 커질수록(즉, 관리하는 데이터가 많아질수록) 프로그램의 관리와 효율성을 위해 활용해야 하는 기법이라고 생각한다.

수업시간에 이제 linked list를 배우고 있는데, 링크트 리스트를 통한 데이터 관리에 대한 것도 (assn4 수준은 아니지만..) 한 번 코딩해보고 싶다.

Case 6:

Printf ========IMAGE LIST==========

Print\_cubic

Printf ========FILTER LIST===========

Print\_cubic

원래는 print\_cubic 내부에 ======%s %s======를 넣고, 인덱스까지 출력할 수 있도록 해서 print\_cubic x 2 만 써넣을라 했으나,

인덱스가 없는 경우 ==========IMAGE LIST=========== 만 출력해야 하기에, 따로 분리했다.