2023 Spring OOP Assignment Report

과제 번호 : 4
학번 : 20220100
이름 : 박기현
Povis ID : kihyun

명예서약 (Honor Code)

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.
I completed this programming task without the improper help of others.

1. 프로그램 개요

□ 본 과제에서는 STL(Standard Template Library)에서 제공하는 기능 중 shared_ptr을 직접 구현해 보고 이를 이용한 영상 처리 클래스를 구현해 봄으로써 template과 STL에 대한 이해도를 높이는 것을 목적으로 한다.

2. 프로그램의 구조 및 알고리즘

□ **SharedPtr**

◆ SharedPtr template 클래스는 shared_ptr의 간소화된 버전의 클래스를 의미한다.

◆ shared_ptr은 <memory> 헤더 파일을 include함으로써 사용할 수 있는데, shared_ptr은 동적 할당된 메모리 영역이 현재 더 이상 사용하지 않는 영역인지를 파악하기 위해 참조 카운트(reference count) 방식을 사용한다.

◆ SharedPtr 생성

- SharedPtr은 template 클래스로, 적어도 하나 이상의 template parameter를 받는다. 만약 MyClass라는 클래스의 객체를 가리키는 포인터를 생성할 경우, *SharedPtr<MyClass> Ptr;* 을 통해 객체를 생성할 수 있다.
- 또한 *SharedPtr<MyClass> ptr(new MyClass());* 와 같이 SharedPtr 객체를 생성함과 동시에 동적 할당된 메모리 영역을 이용하여 초기화할 수 있으며, *SharedPtr<MyClass>ptr(new MyClass); SharedPtr<MyClass> ptr2(ptr);* 과 같이 같은 타입의 객체를 가리키는 다른 SharedPtr 객체를 이용하여 초기화할 수도 있다.

◆ SharedPtr 대입 연산

- SharedPtr 객체를 다른 ShareddPtr 객체에 대입 가능하다. 이때 두 SharedPtr 객체는 공통된 동적 할당된 메모리를 가리키게 된다. 이 경우 참조 카운트가 증가한다.
- SharedPtr 객체에 새로운 동적 할당된 메모리 주소를 바로 대입할 수는 없다. 이미 만들어진 SharedPtr 객체에 새롭게 동적 할당된 메모리 주소를 대입하고자 하는 경우에는 SharedPtr의 생성자를 이용하여 객체를 새로 생성하고 대입해야 한다.

◆ SharedPtr 객체의 이용

- SharedPtr 객체는 자신이 가리키는 객체를 이용할 수 있게 하기 위해 *과 -> 연산자를 지원한다. 이때 두 연산자 모두 const 버전과 non-const 버전을 지원한다.
- SharedPtr 객체는 필요시 일반 포인터로 변환 가능하다.

◆ 자동 동적 할당 해제

- SharedPtr 객체로 더 이상 참조되지 않는 메모리 영역은 자동으로 동적 할당 해제된다.
- SharedPtr 객체는 배열의 동적할당도 지원한다. 배열의 경우 동적 할당을 위해 new와 delete 대신 new[]와 delete[] 연산자를 사용한다. 이를 위해 SharedPtr template은 두 번째 template parameter로 동적 할당 해제를 위한 함수를 입력 받아 일반 객체의 경우와 배열의 경우의 메모리 해제를 다르게 처리한다.

◆ **SharedPtr template 클래스를 객체 지향 프로그래밍이나 제네릭 프로그래밍 관점 또는 다른 측면에서 개선할 수 있다면 어떤 식으로 개선할 수 있을지 기술하라.**

- 객체 지향의 특징으로는 추상화(Abstraction), 캡슐화/정보은닉(Encapsulation/data hiding), 다형성(polymorphism), 상속성(inheritance), 재사용성(reusability)이 있다. SharedPtr 클래스의 대입 연산자 오버로딩에서 SharedPtr과 SharedArray의 경우를 분리하여 구현하였는데, 이때 default parameter를 이용한다면 충분히 재사용성 관점에서 개선할 수 있을 것이라는 생각이 들었다.

□ **Image**

- 이미지 파일 포맷 중 하나인 BMP 파일로부터 이미지를 읽어서 간단한 영상 처리를 거친 후 나온 결과를 BMP 파일로 저장할 수 있는 프로그램이다.
- BMP 파일로 읽은 그림을 문자로 변환하여 출력한다.
- Image 생성
  - Image 클래스 template은 다양한 픽셀 타입을 지원한다. 이때 픽셀 타입은 template parameter로 받는다.
- Image 클래스 template에서 제공하는 public interface
  - 생성자
    - *Image()* – default 생성자
    - *Image(size_t _width, size_t _height)* – 이미지의 크기만큼 메모리를 할당하는 생성자
    - *Image(size_t _width, size_t _height, const PixelType& val)* – 이미지의 크기만큼 메모리를 할당하고 val 값을 이용하여 모든 픽셀값을 초기화하는 생성자
    - *Image(const Image<PixelType>& img)* – 복사 생성자
- 소멸자
  - *~Image()*
- 연산자
  - 대입 연산자 (operator=) – 같은 픽셀 타입을 사용하는 이미지 객체로부터 대입받는 연산자
  - 배열 접근 연산자 (operator[])
- 기타 멤버 함수
  - *size_t width() const* – 이미지 너비 반환
  - *size_t height() const* – 이미지 높이 반환
- Image 클래스의 픽셀값을 저장하기 위한 공간은 동적 할당을 이용한다. 이때 SharedArray를 이용한다.
- **Image template 클래스를 객체 지향 프로그래밍이나 제네릭 프로그래밍 관점 또는 다른 측면에서 개선할 수 있다면 어떤 식으로 개선할 수 있는지 기술하라.**

- 객체 지향의 특징으로는 추상화(Abstraction), 캡슐화/정보은닉(Encapsulation/data hiding), 다형성(polymorphism), 상속성(inheritance), 재사용성(reusability)이 있다. Image template 클래스 또한 마찬가지로, val 값이 주어지지 않았을 때와 val 값이 주어졌을 때의 경우, default parameter를 이용한다면 두 경우의 생성자를 따로 구현하는 것이 아니라, 하나의 생성자로 재사용성을 높일 수 있을 것이라 생각한다.

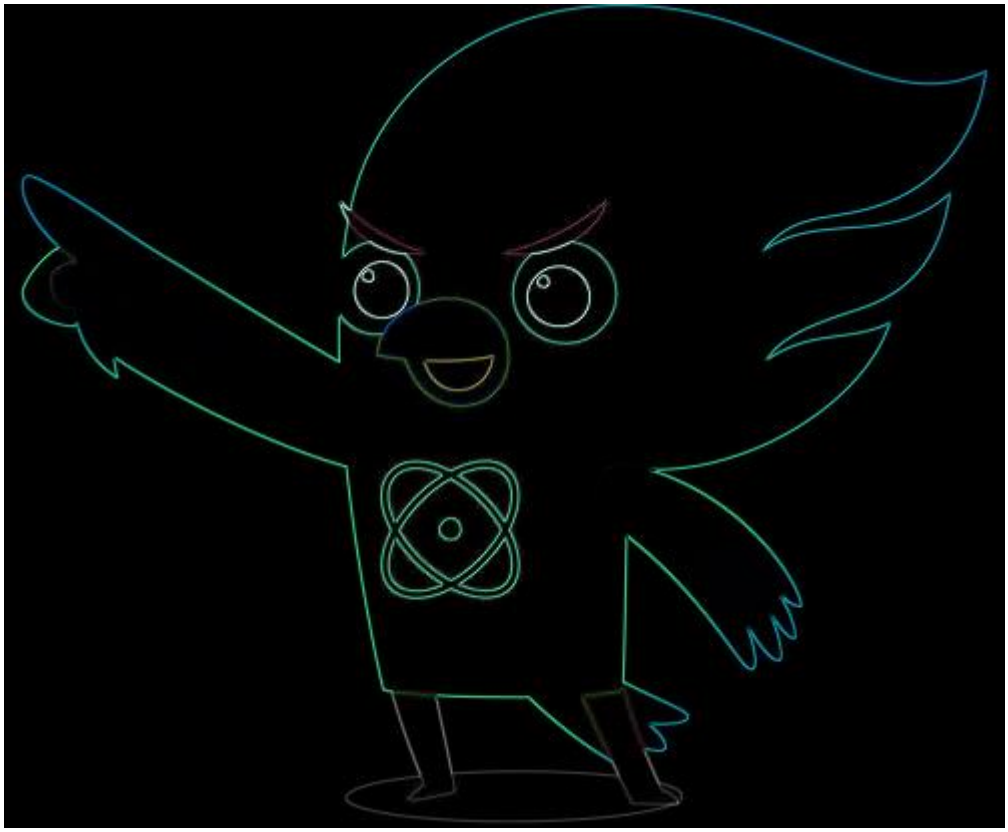
◆ **본인이 구현하고 싶은 다른 영상 처리 예제나 추가 기능이 있다면 구현하고 이를 보고서에 기술할 것**

- compute_edge_map 함수를 변형하여, edge_map을 구하고, r, g, b 별로 x축으로 평행이동하여 글리치 느낌의 영상 처리를 하였다. 출력 결과는 아래에 첨부하였다.

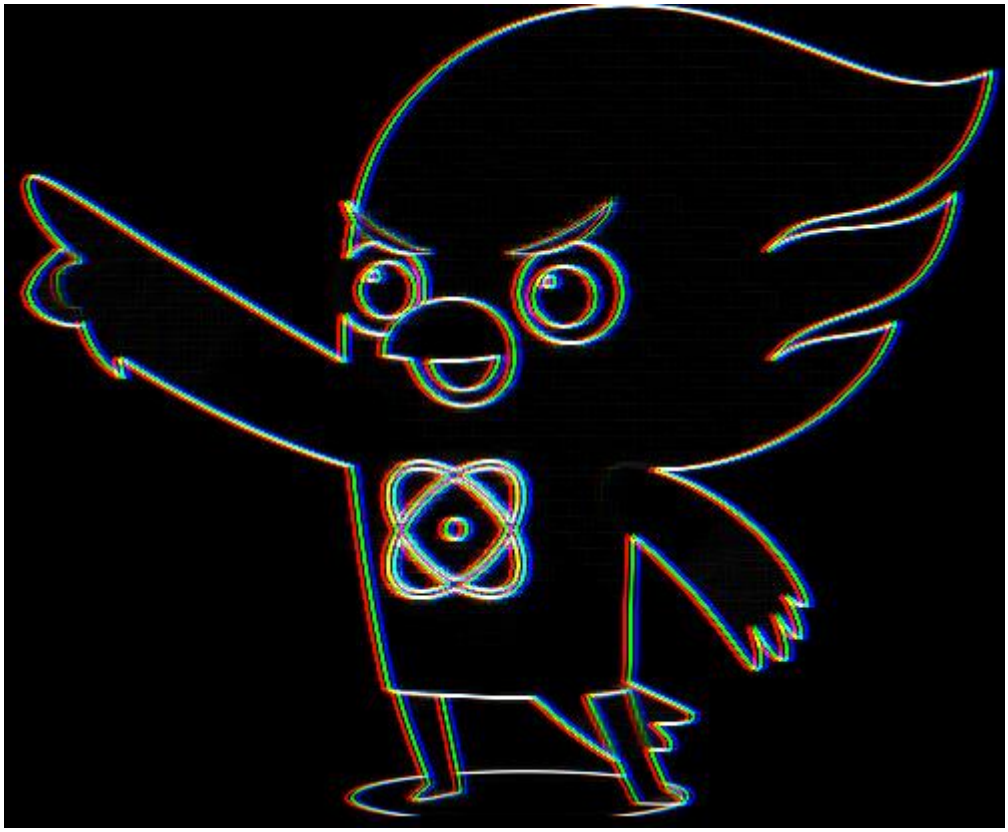
```
< Problem1 test >
test_SharedPtr()
MyClass object(100) created: 1
MyClass object(200) created: 2
=============
ptr1: 200
ptr2: 100
ptr3: 100
=============
Dealloc Object
MyClass object(100) destroyed: 1
=============
ptr1: 200
ptr2: 200
ptr3: 200
=============
MyClass object(300) created: 2
=============
const_ptr: 300
const_ptr: 300
=============
pp: 200
Dealloc Object
MyClass object(300) destroyed: 1
Dealloc Object
MyClass object(200) destroyed: 0

test_SharedArray()
=============
arr1[0]: 1
arr2[0]: 1
arr3[0]: 1
=============
=============
arr1[0]: 2
arr2[0]: 3
arr3[0]: 3
=============
Dealloc Array
=============
arr1[0]: 2
arr2[0]: 2
arr3[0]: 2
=============
Dealloc Array
```

```
< Problem2 test >
Dealloc Array
Dealloc Array
Dealloc Array
Dealloc Array
Dealloc Array
2. ASCII conversion
2.1. Grayscale conversion
Dealloc Array
2.2. Downsampling
Dealloc Array
Dealloc Array
Dealloc Array
2.3. ASCII art drawing
OOOOOOOOOOOOOOOOOOOOOOOOOpEo3Cii}t5SVOOOOOOOOOOOOO
OOOOOOOOOOOOOOOOOOOOOOOde())JJJJJJJ7(uSpOOOOOOOOOO
OOOOOOOOOOOOOOOOOOOOOx7)))))JJJJJJJ777iZhOOOOOOOO
OOOOOOOOOOOOOOOOOOOptv)))))))JJJJJJJ77777Ij6pOd]p
OOOOOOOOOOOOOOOOOOp3vvv)))))))JJJJJJJ77J7777(((CO
OOOOOOOOOOOOOOOOOevvv)v)))))))JJJJJJJ7777777(ZO
OOOOOOOOOOOOOOOOkvvvvvv)))))))))JJJJJJJ77777|9O
OOOOOOOOOOOOOOOOfTvvvvvv)v)))))))JJJJJJJ7777yOO
OwOOOOOOOOOOOOOO6TTTvvvvvvv)))))))JJJJJJJ77YOOO
OFfwOOOOOOOOOOp5TTTvvvvvvv))v)))))))JJJJJJIqVdOO
OZ(7[4OOOOOOOOOVLTTTTvvvvvvT+v)))))))Jfnj6dyidOO
OVI77F2OOOOOOOOOT=zTTTTvvT!^Lv)))))))}Z[3}FJJfOOO
Op3JJJ)19OOOOOOOSk1vTTTTzCjpy)vv)))))7))JJJJJEOOO
Olv7JJJ)J5pOOOOOSz^5TTTT3E/vV[)v)))))))))JJJeOOOO
Ov)JJ))vvTFSOOOOu``[)JTTo/.`xjvvvvv)))))))|yOOOOO
Oy()))vvTTTL19OOd97vtfff7}I`=htvvvvv))))Jt2pOOOOOO
OOpP))vvTLLLsv5pqFo3fffffTuShovvvvvvv))eSaZ3dOOOOO
OOOOe(vTTLLsLLL|ZLvifFF7FJTTTTTTvvvvvv}|)))ZOOOOOO
OOOOpSSFLLssLLLLsLLL|+`.FTTTTTTTvvvvvvv))fpOOOOOO
OOOOOOOpYJLLLLLLLLsLTFJi7LTTTTTTvvvvvvvvf4OOOOOOO
OOOOOOOOVZ)LLLLLLLsLLTLLLTTTTTTTTTvvvvnpOOOOOOOO
OOOOOOOOOOVx|LLLLLLLLLLLLLLTTTTTTTTvv}wOOOOOOOOO
OOOOOOOOOOOOwl)sLLJLs))LLLTTTTTTT(uwOOOOOOOOOOO
OOOOOOOOOOOOOOOpFsl{exfCfLLLLLsJo9OOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOOI)CC{LtJ3LLLLLLLsiqOOOOOOOOOOOOO
OOOOOOOOOOOOOOOOO[LeCLFLl3LLLLLsLLLL5OOOOOOOOOOO
OOOOOOOOOOOOOOOOOYsYTJ5L}1LsLLL7sLsLL[OOOOOOOOOOO
OOOOOOOOOOOOOOOOOETIlLs7f3LLLLLPIsLTTvYOOOOOOOOOO
OOOOOOOOOOOOOOOOO9TITlIfs1LLsss9p3LTvvJkOOOOOOOOO
OOOOOOOOOOOOOOOOOv{I3f337LLLLLdOpIvv)JuOOOOOOOOO
OOOOOOOOOOOOOOOOO{ssLsLLLssLLs4OOV{)J(EOOOOOOOOO
OOOOOOOOOOOOOOOOOuLsLsLLLLLLLsVOOO6(C3EOOOOOOOOO
OOOOOOOOOOOOOOOOOjLLLLLLLLLLLspOOOO]o5OOOOOOOOOO
OOOOOOOOOOOOOOOOOPsLLLLLLLLLLLpOOOOO4OOOOOOOOOOO
OOOOOOOOOOOOOOOOOOl)iultt|ss)JiypOOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOOOOk|fOOOO4}LJ(J7fVOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOOOOOC|VOOOOpnv((3pOOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOOOOO5|qOOOOOOSC(apOOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOOOOVh2(e22]]2ESx|oOOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOOOE5ZlI[55555555CF5OOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOOOphqE]ayjjjya2Sk9OOOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOOOOOOOOOpppOOOOOOOOOOOOOOOOOOOOOOO
Dealloc Array
Dealloc Array
Dealloc Array
Dealloc Array
Dealloc Array
```



edge.bmp



mystyle.bmp

3. 토론 및 개선

- □ 이번 과제를 통해 smart pointer 중 shared_ptr의 개념에 대해 이해할 수 있었고, 이를 단지 헤더 파일에 포함시켜 사용하는 것이 아니라 클래스로 간소화된 버전의 shared_ptr을 직접 구현해봄으로써 작동 원리와, 참조 카운팅에 대해서 구체적으로 학습할 수 있었다.

- □ 이번 과제에서 어려웠던 점은 문제 1에서 구현했던 SharedArray를 이용하여 문제 2에서 Image의 동적 할당에 사용해야 한다는 점이었는데, 처음에는 Image 객체를 어떤 타입으로 선언해야하는지, 또한, 그 선언한 객체에 너비와 높이에 맞는 배열을 어떻게 동적 할당해야하는지 감이 잡히지 않아 많이 헤맸었던 것 같다. 이 과정에서 template과 SharedPtr의 동작 원리를 하나씩 따라가보는 과정을 통해 해결할 수 있었다.

4. 참고 문헌

- □ 조성현 교수님의 <객체지향프로그래밍> 수업 자료