**2023 Spring OOP Assignment Report**

과제 번호 : 4

학번 : 20190445

이름 : 허수범

Povis ID : sbh408

**명예서약 (Honor Code)**

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

I completed this programming task without the improper help of others.

프로그램을 하다 보면 결정해야 할 세부 사항이 많은데, 이러한 세부 사항을 처리한 방법과 이유를 보고서에 쓰십시오.

독창적인 아이디어와 추가 기능은 보너스 점수를 받을 수 있으므로, 보고서에 명확히 기재하십시오.

문제가 여러 개인 경우, 각 문제별로 정리해서 작성합니다.

아래 문항별 설명은 편의를 위한 것으로, 삭제하고 제출한다.

**문제 1 - SharedPtr**

1. **프로그램 개요**

문제 1에서 요구한 것은 기존에 STL에서 제공되던 shared\_ptr의 간소호된 버전인 SharedPtr를 구현하는 것이었다. 스마트 포인터의 종류에는 여러가지가 있는데, 본 문제에서는 참조 카운트 방식을 사용하는 shared\_ptr를 구현하였다. 이 외의 제약조건은 Assignment4.pdf의 1.2 과제 요구사항에 맞춰 스마트 포인터를 구현하였다. 프로그램의 동작은 모두 SharedPtr\_image\_test.cc 내에서 이루어지므로 따로 사용자가 동작을 입력할 필요 없다.

프로그램 디렉토리 내에는 작성한 헤더 파일의 작동을 test할 수 있게 만들어진 SharedPtr\_image\_test.cc 프로그램 소스 코드, 소스코드들의 컴파일과 실행, 디렉토리 정리 등을 간편히 실행할 수 있도록 만들어주는 Makefile, 문제2의 Image processing을 구현한 Image.cc와 Image.h 소스 코드, image processing을 적용할 input.bmp파일(그림), 그리고 본 문제가 구현되어 있는 SharedPtr.h 파일이 있다.

1. **프로그램의 구조 및 알고리즘**

본 문제에서는 주어진 SharedPtr.h내의 지정된 부분에서 SharedPtr class의 멤버들을 구현하였다.

explicit SharedPtr(ObjectType\* ptr = nullptr):

ObjectType의 포인터를 argument로 받는 생성자이다. 앞의 explicit 키워드는 암시적 형변화를 막아 예상치 못한 동작을 예방하기 위해 붙였고, argument는 nullptr를 default argument로 지정하여 default constructor처럼 작동할 수 있게 하였다. 생성시 initializer list를 통해 입력받은 포인터를 SharedPtr object의 멤버 변수 m\_object에 대입하고, m\_ref\_counter, 즉 참조 카운터 포인터를 nullptr로 초기화하도록 하였다. 이후 생성자 만약 argument로 받은 포인터가 nullptr가 아닌 경우, 참조 카운터 포인터에 new int(1)를 동적할당 하도록 하였다.

SharedPtr(const SharedPtr<ObjectType>& other):

SharedPtr를 위한 copy constructor이다. 같은 class의 object의 reference를 argument로 받으며, initializer list를 이용해 argument로 받은 object의 멤버 변수들의 값을 this object의 멤버 변수들에 대입하도록 하였다. 이후 copy 생성자 내에서는 대입해서 받은 m\_ref\_counter, 즉 참조 카운터를 향한 포인터가 nullptr가 아닌 경우 dereferencing하여 1만큼 increment하도록 하였다.

SharedPtr(const SharedPtr<ObjectType, ArrayDeallocator<ObjectType>>& other):

이는 두번째 template argument의 default argument가 있기 때문에 SharedArray를 위해 따로 정의해준 복사 생성자이다. 동작은 위의 복사 생성자와 동일하다.

~SharedPtr():

밑에서 설명할 reset함수를 호출한다.

void reset():

참조 카운터가 nullptr가 아닌 경우, 참조 카운터를 1만큼 decrement하고, decrement한 결과 0이 되었으면, Dealloc(m\_object), 즉 클래스 멤버 변수 중 객체에 해당하는 부분을 함수 포인터를 이용하여 할당 해제하고, 참조 카운터 포인터를 할당 해제하도록 해주는 함수이다. 위의 소멸자에서도 호출되지만, 대입 연산자의 overloading에도 이용된다.

SharedPtr<ObjectType>& operator=(const SharedPtr<ObjectType>& other):

Assignment operator로, SharedPtr을 argument로 받는다. 만약 argument와 this object가 같은 주소를 가지지 않는다면(다른 객체라면), reset()을 호출한 후, 각 멤버 변수를 대입해준 후, m\_ref\_counter가 널포인터가 아닌 경우 1만큼 increment해주도록 하였다. 이후 \*this를 return해준다.

SharedPtr<ObjectType, ArrayDeallocator<ObjectType>>& operator=(const SharedPtr<ObjectType, ArrayDeallocator<ObjectType>>& other):

이는 두번째 template argument의 default argument가 있기 때문에 SharedArray를 위해 따로 정의해준 대입 연산자이다. 동작은 위의 대입 연산자와 동일하다.

objectType\* operator->(), const ObjectType\* operator->() const:

const와 비-const객체를 위한 -> 연산자 오버로딩이다. m\_object를 return한다.

objectType& operator\*(), const ObjectType& operator\*() const:

const와 비-const객체를 위한 \* 연산자 오버로딩이다. \*m\_object를 return한다.

objectType& operator[](int index), const ObjectType& operator[](int index) const:

SharedArray를 위한 []연산자로, 각 배열의 각 원소에 접근하도록 해주는 연산자이다. Const와 비-const객체를 위해 두 개의 함수를 정의하였다. m\_object[index]를 return해준다.

operator ObjectType\*(), operator ObjectType const\*() const:

const와 비-const객체를 위한 typecasting operator이다. m\_object를 return한다.

1. **토론 및 개선**

생성자, 복사 생성자, 대입 연산자, 등을 이용하여, 객체 내에서 동적 할당이 일어나더라도 정상적으로 작동할 수 있도록 프로그래밍하는 법을 배운 것 같다. 메모리의 동적 할당과 해제가 원래 굉장히 헷갈리는 개념이었는데, 이 문제를 해결하며 이해가 깊어진 느낌을 받았다.

객체지향 프로그래밍의 관점에서 보았을 때, 제대로 하지 못해서 아쉬운 점은, 다양한 overloading을 통한 다형성을 십분 활용하지 못했던 것이다. Array와 그냥 변수에 해당하는 함수들을 각가 한번씩 정의할 필요 없이, 한번에 정의할 수 있는 방법들이 존재할 텐데, 하지 못해 아쉽다. 제네릭 프로그래밍의 관점에서 보았을 때, 구현한 결과는 어떤 변수, 심지어 배열, 고차원의 배열이 입력되어도 작동할 수 있도록 하였기 때문에(이는 Image가 정상적으로 작동하는 데에서 볼 수 있다.) 잘 구현되었다고 생각한다.

1. **참고 문헌**

<http://www.tcpschool.com/cpp/cpp_template_smartPointer> 스마트 포인터에 대한 깊은 이해를 위해

<https://nobilitycat.tistory.com/entry/Reference-Counting> reference counting

**문제 2 – Image**

1. **프로그램 개요**

문제 2번에서는 문제1에서 구현한 SharedArray를 이용해 Image class를 정의하고, 이를 이용해 간단한 이미지 프로세싱을 해보았다. Image class는 너비 m\_width와 높이 m\_height를 size\_t형 변수로 가지고, 픽셀값들을 SharedArray형 배열 m\_buff로 갖도록 하였다. SharedPtr\_image\_test.cc를 이용해 이미지 프로세싱을 할 수 있는데, .bmp파일을 입력받아 이미지 프로세싱을 한 후 .bmp파일로 출력할 수 있고, ASCII art로도 표현하여 console에 띄울 수 있도록 되어있다.

디렉토리 내의 파일들에 대한 설명을 문제1에서 하였으므로 생략한다.

1. **프로그램의 구조 및 알고리즘**

본 문제에서는 주어진 Image.h내의 지정된 부분에서 Image class의 멤버들을 구현하였다. 멤버 변수들은 위에서 설명하였다.

생성자와 소멸자:

Default constructor는 initializer list로 m\_width와 m\_height를 0으로 초기화하고, m\_buff를 nullptr로 하도록 하였다. 두 번째 constructor는 너비와 높이를 argument로 받도록 하였다. 역시 초기화 리스트로 너비와 높이를 대입해주었고, m\_buff는 SharedArray의 생성자를 이용해 배열을 동적으로 할당해 주었다. 세번째 constructor는 두번째와 같지만, 세번째 argument로 const PixelType& val를 받는다. m\_buff에 배열을 동적으로 할당해준 후, for문을 너비\*높이만큼 iterate하도록 해 m\_buff의 각 원소를 val로 초기화해주도록 하였다. 복사 생성자는 너비와 높이는 초기화 리스트로 그대로 받지만, m\_buff에 배열을 스마트 포인터로 동적할당해준 후, for문을 이용해 각 원소에 대상이 되는 객체의 각 원소를 대입하여 deep copy가 되도록 하였다. 소멸자는 특별한 일을 하지 않아도 되도록 하였다.

Assignment operator overloading:

대입 연산자의 경우, this 객체와 other 객체가 다른 경우(두 개겣를 가리키는 포인터가 다른 값을 가지면), m\_width와 m\_height를 같도록 해주고, m\_buff는 복사 생성자와 동일하게 deep copy해주도록 하였다. 이후 \*this를 return하도록 해주었다.

size\_t width() const, size\_t height() const:

private member인 너비와 높이에 접근하기 위한 함수로, 해당하는 멤버 변수를 각각 return하도록 해주었다.

1. **토론 및 개선**

가장 크게 실감했던 것은 스마트 포인터의 효용성이었다. 아주 편리하게 메모리를 걱정하기 않고 포인터를 사용할 수 있었으며, 어떤 타입이든, 어떤 차원이든 상관없이 응요할 수 있어 매우 편해 제네릭 프로그래밍의 관점에서도 아주 잘 구현되었다고 생각한다. 하지만, 픽셀값의 다양성이 커도 같은 클래스 내에서는 잘 작동하지만, 다양한 image processing을 구현할 때에 불편한 점이 없잖아 있었다. 다양한 타입들 간에도 연산을 할 수 있는 멤버 함수나 추가적인 클래스, 추가적인 상속 클래스 등을 선언하면 더욱 제너릭한 프로그램이 되지 않을까 생각한다.

A picture containing sky, outdoor, tree, landscape

Description automatically generated

**A picture containing text, screenshot, font

Description automatically generated**

**A blurry image of a field of flowers

Description automatically generated with low confidence**

**A picture containing text, screenshot, font

Description automatically generated**

**A picture containing sky, landscape, night, tree

Description automatically generated**

**A screen shot of a computer code

Description automatically generated with low confidence**

**A picture containing sky, night, tree, star

Description automatically generated**

가장 위의 사진은 주어진 이미지가 아닌 다른 이미지로 새로운 이미지 프로세싱을 시험해보기 위해 가져온 bmp이미지이다. 각 코드들은 test.cc에 작성되어있으며, 이로 인해 프로그램을 실행하면 나와야 하는 결과보다 더욱 많은 deallocation이 일어나는 것을 볼 수 있다.

구현한 average filter는 설정한 필터의 크기만큼 인접 픽셀들의 값을 평균내어 mapping하도 록 하였고, 추가호 bilateral filter역시 구현해 보았다. Pa3에서 이미 구현해보았던 코드들을 그대로 사용하였다. 아래 참고 문서 등을 참고하여 경계선을 유지한 채 흐릿해지는 필터를 구현해보려 한 결과이고, 상당히 잘 나온 결과라고 판단된다. Spin을 하도록 하는 함수도 구현해 보았고, 이 역시 잘 작동하는 것을 볼 수 있다.

1. **참고 문헌**

<https://dsp.stackexchange.com/questions/61988/understanding-the-bilateral-filter-neighbors-and-sigma> bilateral filter 설명