**2023 Spring OOP Assignment Report**

과제 번호 : #4

학번 : 20220923

이름 : 이윤혁

Povis ID : leeyoonhyuk0

**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.

1. **프로그램 개요**

이 프로그램은 STL에 포함되어 있는 Smart pointer인 shared\_ptr의 일부 기능을 직접 구한 SharedPtr class를 가지고 있고, 구현한 Smart pointer로 Image Class를 만들어 다양한 영상처리를 한다. 프로그램 디렉토리는 [표 1]과 같이 구성되어있다.

[표 1] 프로그램 디렉토리

|  |  |
| --- | --- |
| 파일명 | 설명 |
| Image.cc | Image Class를 조작할 수 있는 function들이 정의되어 있는 소스파일이다. |
| Image.h | Image Class의 헤더파일, Class가 선언 및 정의되어 있다. |
| SharedPtr.h | SharedPtr, SharedArray Class의 헤더파일, Class가 선언 및 정의되어 있다. |
| SharedPtr\_Image\_test.cc | SharedPtr과 Image Class의 정상작동을 확인하기 위해 테스트하는 소스파일이다. |

1. **프로그램의 구조 및 알고리즘**
   1. SharedPtr

SharedPtr는 STL의 shared\_ptr의 일부 기능을 구현한 smart pointer class이다. SharedPtr은 contain 할 Object의 type을 설정할 수 있다. 모든 SharedPtr가 특정 Object를 더 이상 가리키지 않으면, 자동으로 그 Object를 할당 해제해준다. Assignment operator로 일반 포인터를 SharedPtr에 넣지 못하도록 일반 포인터를 받는Assigment operator를 = delete 구문으로 제거했다. [표 2]는 SharedPtr의 method를 설명한다. 템플릿(template<typename ObjectType, DeallocatorFuncType<ObjectType> Dealloc = Deallocator<ObjectType> >)은 너무 길어서 표에서 생략한다. [표 3]은 SharedPtr의 member variable를 설명한다.

[표 2] Method of SharedPtr

|  |  |  |
| --- | --- | --- |
| 접근 제한자 | {반환형}{함수명}({파라미터명}) | 설명 |
| public | SharedPtr() | Default constructor |
| public | SharedPtr(ObjectType\*object\_) | Constructor, ObjectType을 pointer가 생성될 때 바로 받는다. |
| public | explicit SharedPtr(const SharedPtr<ObjectType, Dealloc>& original\_ptr) | Copy constructor, 같은 자료형인 SharedPtr를 생성될 때 바로 받는다. |
| public | ~SharedPtr() | Destructor, 파괴 |
| public | SharedPtr<ObjectType, Dealloc>& operator=(const SharedPtr<ObjectType, Dealloc>& original\_ptr) | Assignment Operator, 같은 자료형인 SharedPtr를 대입연산을 통해 받는다. |
| public | ObjectType& operator \*() | Dereference Operator(non-const version) |
| public | const ObjectType& operator \*() const | Dereference Operator(const version) |
| public | ObjectType\* operator ->() | Arrow Operator(non-const version) |
| public | const ObjectType\* operator ->() const | Arrow Operator(const version) |
| public | ObjectType& operator[](int i) | Subscript Operator(non-const version) |
| public | const ObjectType& operator[](int i) const | Subscript Operator(const version) |
| public | operator ObjectType\* () | Type cast Operator(non-const version) |
| public | operator ObjectType const\* () const | Type cast Operator(const version) |

[표 3] Member variable of SharedPtr

|  |  |  |
| --- | --- | --- |
| 접근 제한자 | {자료형}{변수명} | 설명 |
| private | ObjectType\* m\_object | SharedPtr로 가리킬 object의 pointer |
| private | int \*m\_ref\_counter | 현재 object를 가리키고 있는 pointer의 개수의 pointer |

. 문서의 요구사항을 충족하기 위해 Assignment operator로 일반 포인터를 SharedPtr에 넣지 못하도록 일반 포인터를 받는 Assigment operator를 =delete 구문으로 제거할 수 있다.

예시상황

|  |
| --- |
| SharedPtr<MyClass> ptr;  ptr = new MyClass(); |

이 경우 다음과 같은 컴파일 에러가 발생한다.



Assignment Operator를 =delete 하는 방법 이외에도 ObjectType\*를 parameter로 받는 Constructor에 explicit 구문을 넣으면 예시상황에서도 다음과 같은 컴파일 에러가 발생한다.



Implicit Typecasting으로 Contstructor에 값이 입력되기 때문이다.

제출한 코드는 Assignemt Operator를 제거하지 않고, Constructor를 explicit하게 설정했다.

* 1. SharedArray

SharedArray는 SharedPtr로 구현한 smart pointer로, Array를 동적할당 할 수 있다는 점 이외에는 SharedPtr와 동일하다. [표 4]는 SharedArray의 method를 설명한다. 템플릿(template<typename ObjectType, DeallocatorFuncType<ObjectType> Dealloc = Deallocator<ObjectType> >)은 너무 길어서 표에서 생략한다. [표 5]는 SharedArray의 member variable를 설명한다.

[표 4] Method of SharedArray

|  |  |  |
| --- | --- | --- |
| 접근 제한자 | {반환형}{함수명}({파라미터명}) | 설명 |
| public | SharedPtr() | Default constructor |
| public | SharedPtr(ObjectType\*object\_) | Constructor, ObjectType을 pointer가 생성될 때 바로 받는다. |
| public | explicit SharedPtr(const SharedPtr<ObjectType, Dealloc>& original\_ptr) | Copy constructor, 같은 자료형인 SharedPtr를 생성될 때 바로 받는다. |
| public | ~SharedPtr() | Destructor, 파괴 |
| public | SharedPtr<ObjectType, Dealloc>& operator=(const SharedPtr<ObjectType, Dealloc>& original\_ptr) | Assignment Operator, 같은 자료형인 SharedPtr를 대입연산을 통해 받는다. |
| public | ObjectType& operator \*() | Dereference Operator(non-const version) |
| public | const ObjectType& operator \*() const | Dereference Operator(const version) |
| public | ObjectType\* operator ->() | Arrow Operator(non-const version) |
| public | const ObjectType\* operator ->() const | Arrow Operator(const version) |
| public | ObjectType& operator[](int i) | Subscript Operator(non-const version) |
| public | const ObjectType& operator[](int i) const | Subscript Operator(const version) |
| public | operator ObjectType\* () | Type cast Operator(non-const version) |
| public | operator ObjectType const\* () const | Type cast Operator(const version) |

[표 5] Member variable of SharedArray

|  |  |  |
| --- | --- | --- |
| 접근 제한자 | {자료형}{변수명} | 설명 |
| private | ObjectType\* m\_object | SharedPtr로 가리킬 object의 pointer |
| private | int \*m\_ref\_counter | 현재 object를 가리키고 있는 pointer의 개수의 pointer |

* 1. Image

Image는 Pixel을 표현할 다양한 종류(int\_32, float, RGBf, RGB8b)의 값을 2차원 배열로 받을 수 있는 이미지 클래스이다. Copy Constructor와 Assignment Operator로 복사 또는 대입이 일어나는 경우, Deep copy를 수행한다. [표 6]은 Image의 method이다. [표 7]은 Image의 member variable이다.

[표 6] Method of Image

|  |  |  |
| --- | --- | --- |
| 접근 제한자 | {반환형}{함수명}({파라미터명}) | 설명 |
| Public | Image() | Default constructor |
| Public | Image(size\_t \_width, size\_t \_height) | Constructor, Image의 width와 height를 입력받는다. |
| Public | Image(size\_t \_width, size\_t \_height, const PixelType& val) | Constructor, Image의 width, height, Pixel 초기값 val을 입력받는다. |
| Public | explicit Image(const Image<PixelType>& Img) | Copy Constuctor, Image<PixelType>& img를 입력받아 Deep copy를 수행한다. |
| Public | ~Image() | Default Destructor |
| Public | Image<PixelType>& operator=(const Image<PixelType>& img) | Assignment Operator, Image<PixelType>& img를 입력받아 Deep copy를 수행한다. |
| Public | PixelType\* operator[](int y) | Subscript Operator(non-const version) |
| Public | const PixelType\* operator[](int y) const | Subscript Operator(const version) |
| Public | size\_t width() const | Width의 accessor |
| Public | size\_t height() const | Height의 accessor |

|  |  |  |
| --- | --- | --- |
| 접근 제한자 |  |  |
| private |  |  |
| private |  |  |
| private |  |  |

1. **토론 및 개선**

이번 과제를 통해서 Class Templete의 사용 및 Smart pointer의 작동방식을 숙지할 수 있었다. 또한 Consturctor, Destructor, Copy Constructor, Assignment operator를 overloading 하면서 class의 작동 방식을 익힐 수 있었다.

SharedPtr\_image\_Test.cc에 Image를 조작하는 function을 새로 추가했다. function을 test하는 main()은 기존 main() 아래에 주석처리 해 놓았다.

[코드 1] Color inversion function

|  |
| --- |
| void compute\_inversioned\_img(const Image<RGB8b>& img, Image<RGB8b>& inversioned\_img){  /\*  \* The function that performs color inversion of given image  \*  \* :param const Image<RGB8b>& img: The given image  \* :param Image<RGB8b>& inversion\_img: The image result of given image's color inversion  \*  \* :return void:  \*/  inversioned\_img = Image<RGB8b>(img.width(), img.height(), RGB8b(0, 0, 0));  // for every channel in every pixel  for (int y = 0; y < img.height(); y++) {  for (int x = 0; x < img.width(); x++) {  for (int ch = 0; ch < 3; ch++) {  inversioned\_img[y][x][ch] = 255 - img[y][x][ch]; // Color inversion  }  }  }  } |

[코드 2] Alpha blending function

|  |
| --- |
| void compute\_alpha\_blended\_img(const Image<RGB8b>& img, Image<RGB8b>& alpha\_blended\_img, float alpha) {  /\*  \* The function that performs alpha blending of given image  \*  \* :param const Image<RGB8b>& img: The given image  \* :param Image<RGB8b>& inversion\_img: The image result of given image's alpha blending  \* :param float alpha: alpha value(transparency)  \* :return void:  \*/  alpha\_blended\_img = Image<RGB8b>(img.width(), img.height(), RGB8b(0, 0, 0));  // for every channel in every pixel  for (int y = 0; y < img.height(); y++) {  for (int x = 0; x < img.width(); x++) {  for (int ch = 0; ch < 3; ch++) {  alpha\_blended\_img[y][x][ch] = int(img[y][x][ch] \* alpha + 255 \* (1 - alpha)); // Alpha blend image with white background  }  }  }  } |

[그림 1]은 주어진 Ponix bmp 사진을 통해 Color inversion을 한 결과이다.

[그림 1] Color inversioned ponix



[그림 2]는 주어진 Ponix bmp 사진을 통해 Alpha 값을 0.5로, 하얀 배경화면에 Alpha blending한 결과이다.

[그림 2] Alpha blended ponix



Image class는 copy constructor나 assignment operator에서 Deep copy를 수행하는데, SharedPtr은 shallow copy를 하기에 더 적절한 pointer이다. 여러 pointer가 한 object를 가리킬 수 있기 때문이다. Image의 m\_buff를 unique\_ptr처럼 한 object는 한 pointer가 가리킬 수 있는 smart pointer를 사용했으면, 사용한 pointer가 좀 더 목적에 부합했을 것이다. 물론 SharedPtr, unique\_ptr 모두 smart pointer로써 소멸시 자동으로 동적할당 해제를 하므로, 두 포인터 모두 자동 동적할당 해제의 목표에는 부합한다.

1. **참고 문헌**

C++ reference : https://en.cppreference.com/w/

2023S object oriented programming slides, Appendix of Assignment4 document

Microsoft learn c++ : https://learn.microsoft.com/en-us/cpp/?view=msvc-170

Alpha blending Wikipedia: https://en.wikipedia.org/wiki/Alpha\_compositing