2023 Spring OOP Assignment Report

과제 번호 : 4
학번 : 20220302
이름 : 김지현
Povis ID : jihyunk

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

- ◻ 과제 1번에서는 smart pointer 종류 중 하나인 shared pointer 를 구현하는 역할을 할 것이다.
- ◻ Shared pointer 에서는 template 을 이용해 임의의 객체 타입에 대해서 자동으로 동적할당을 하고 프로그램이 종료 되거나 더이상 그 pointer 가 사용되지 않으면 동적 할당 해제를 해주는 역할을 한다.
- ◻ Shared pointer class function 으로는 생성자와 소멸자, 대입 연산자(operator =), 객체의 이용 (* 또는 ->), 마지막으로 동적 할당된 배열의 지원 ([ ]) 등이 있다.
- ◻ 과제 2번에서는 과제 1 에서 만든 shared pointer 를 이용해 이미지 처리를 위한 템플릿 클래스를 작성한다.
- ◻ Image class 에서 각 픽셀값을 저장하는 RGBf 타입을 가진 Shared pointer 와 픽셀의 가로, 세로 값을 저장하는 m_width 와 m_height 을 이용해 생성자, 소멸자, 연산자, 그리고 기타 멤버 함수를 작성했다.
- ◻ 프로그램을 실행하기 위해서는 bmp 타입을 가지는 input file 이 있어야 하며 실행을 했을 때, 영상 처리를 거친 후의 결과 bmp 파일이 "edge.bmp" 로 저장이 될 것이고, 터미널에는 input file 을 문자로 바꾼 결과가 프린트 될 것이다.

2. 프로그램의 구조 및 알고리즘

SharedPtr, typename ObjectType	
Private	Public
ObjectType* m_object int* m_ref_counter	SharedPtr (ObjectType* object = NULL) SharedPtr (const SharedPtr <ObjectType, Dealloc>& other) ~SharedPtr()

	SharedPtr& operator = (const SharedPtr& other) ObjectType* operator -> () const ObjectType* operator -> () const ObjectType& operator* () const ObjectType& operator* () const ObjectType & operator [] (int index) const ObjectType & operator [] (int index) const operator ObjectType const* () const operator ObjectType* ()

- m_object is a private member pointer with template of ObjectType.
- m_ref_counter is a integer type pointer which counts the number of pointer that counts the same memory.
- SharedPtr (ObjectType* object = NULL) is a default constructor. When constructing the object of class SharedPtr, and object is not declared, then the “object” will be a null pointer. Then assign object to m_object and m_ref_counter to null pointer. If object is not a null pointer, dynamically allocate m_ref_counter and set equal to 1.
- SharedPtr (const SharedPtr <ObjectType, Dealloc>& other) is a copy constructor. When declaring “SharedPtr<Object Type> ptr2 (ptr1);”, in cpp file, ptr 2 will points the memory ptr 1 pointed to. Therefore, in the function definition, each m_object and m_ref_counter will be “copied” and if the ptr 1 is not a null pointer (this is the if statement) the m_ref_counter will be added by one.
- ~SharedPtr() is a destructor. If the pointer exists and the reference counter equals to 1 (meaning only one pointer points to that certain memory), then run the deallocation function, called “Dealloc”, for m_object and delete m_ref_counter.
- SharedPtr& operator = (const SharedPtr& other) is an assignment operator. If the “this” pointer, which is a reference to an invoking object, equals to the “other” shared pointer, than the function returns nothing. If not, the check the m_ref_counter and if exists and equals to 1, Dealloc m_object and delete m_ref_counter. Then, set m_object and m_ref_counter each equals to “other” pointer’s m_object and m_ref_counter. Then, if the m_ref_counter is not a null pointer, add 1 to m_ref_counter.
- ObjectType* operator -> () is a pointer operator for “->”. This function will return m_object.
- const ObjectType* operator -> () const does the same function as above but this

is a constant function, meaning that the function will not modify inside the function and returns a constant value / pointer.

- ObjectType& operator* () is a pointer operator for "*". This function will return *m_object.
- const ObjectType& operator* () const does the same function as above but this is a constant function.
- ObjectType & operator [] (int index) accesses an array element. It returns m_object[index].
- const ObjectType & operator [] (int index) const does the same function as above but this is a constant function.
- operator ObjectType* () is a type casting operator. It is used when converting the Shared pointer to regular pointer. For Instance, it is used when assigning shared pointer ptr1 of type MyClass to a regular pointer MyClass* ptr2.
- operator ObjectType const* () const does the same function but for constant regular pointer.

Image, typename PixelType	
Private	Public
SharedArray <PixelType> m_buff size_t m_width size_t m_height	Image () Image (size_t _width, size_t _height) Image (size_t _width, size_t _hieght, const PixelType& val) Image (const Image<PixelType>& img) ~Image () Image& operator = (const Image <PixelType>& img) const Image& operator = (const Image <PixelType>& img) const size_t width() const size_t height () const

- m_buff is a private member of class Image and an object of shared pointer with object type as pixel type, another template for class Image.
- m_width is a private member of class Image that saves the width of the image. It is size_t data. Size_t is frequently used data type for sizes or length in STL.
- m_height is a private member of class Image that saves the height of the image.
- Image () is a default constructor. This constructor sets a shared pointer m_buff to null pointer and m_width and m_height to 0.

- ◻ Image (size_t _width, size_t _height) is a constructor when width and height are given. With the given width and height, m_buff is dynamically allocated with array size of width times height.
- ◻ Image (size_t _width, size_t _hieght, const PixelType& val) is another constructor when width, height, and the PixelType value is given as a parameter. Do the same thing as above constructor and additionally, set m_buff [0] ~ m_buff[width * height] to val using for loop.
- ◻ Image (const Image<PixelType>& img) is a copy constructor. In this function, set m_width and m_height to img.m_width and img.m_height, accordingly. Then dynamically allocate m_buff of an array size width * height. Then set m_buff [0] ~ m_buff[width * height] to img [0] ~ img [width * height], accordingly using for loop.
- ◻ ~Image () is a destructor. Since m_buff is a shared pointer, it will automatically delete itself. Therefore, the destructor doesn't have function definition.
- ◻ Image& operator = (const Image <PixelType>& img) is a assignment operator. If "this" pointer doesn't equals to "img" pointer, dynamically allocate m_buff to an array size of width of img * height of img. Then set m_buff [0] ~ m_buff[width * height] to img [0] ~ img [width * height], accordingly using for loop. Finally, set m_with and m_height to width and height of image.
- ◻ const Image& operator = (const Image <PixelType>& img) const does the same function as above, but it is a constant function.
- ◻ size_t width() const returns the width of the image (m_width). Since m_width has a data type of size_t, so its return type is also size_t. Also, the function does not modify any values, so it's a constant function.
- ◻ size_t height () const returns the height of the image (m_height).

3. 토론 및 개선

- ◻ From this assignment, I believe I've gained a complete understanding of the concept of smart pointer, especially shared pointer. While working on the assignment, I have learned the difference between different kinds of smart pointers. Also, I have learned how to make constructors depending on different parameters.
- ◻ Furthermore, I could learn how to apply smart pointers through the second assignment. And I have learned "size_t" data type in class but I have never used. So, I had to first understand the data type, then use it in this code. Therefore, I could gain knowledge on the data type.
- ◻ Also, while understanding the cpp file, I could learn about the 8 bit unsigned

integer type and a way to manipulate image.

- ◻ However, if I had to write cpp file by myself, it would take much more time. So, I think I should work on examining the cpp code also.
- ◻ Based on the codes in image cc, there could be other programs that manipulate the input image file, such as color changes.
- ◻ 실행속도를 높이기 위해서는 shared pointer 보다는 unique pointer 를 사용하는 것이 더 나을 것이다. Shared pointer 는 여러 pointer 가 같은 메모리를 가리킬 수 있지만 그만큼 메모리를 많이 사용하기 때문에 실행속도가 늦어진다. 만약 구현하고자 하는 프로그램이 시간에 예민하다면, 하나의 객체를 하나의 포인터만 가리켜야 하는 불편함을 감수하더라도 unique pointer 를 사용하는 것이 실행 속도 방면에서는 이득일 것이다.

4. 참고 문헌

- ◻ “nullptr” -> https://www.geeksforgeeks.org/understanding-nullptr-c/