**2023 Spring OOP Assignment Report**

과제 번호 : 4

학번 : 20220124

이름 : 김문겸

Povis ID : kkomy

**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.

**1번문제**

1. **프로그램 개요**
   * 해당 프로그램은 스마트 포인터의 일종인 Shared\_Ptr의 기능을 구현해보는 것이다. 우리가 일반적으로 알고 있는 포인터는 동적할당을 해주었다가, 직접 수동으로 동적할당을 해주어야 했다. 하지만 스마트 포인터는 자동으로 메모리를 동적해제 해줌으로써 편리성을 제공한다. 이러한 스마트 포인터의 일종인 Shared\_ptr은 동적할당이 된 메모리가 사용하는지 여부를 참조카운트 방식을 통해 구분하는데, 어떤 포인터가 해당 메모리를 참조할 때마다 count를 1씩 늘여가고, 참조하고 있던 포인터가 할당해제되면 count를 -1해준다. 이렇게 하다가 count가 0이 되면 해당 메모리를 할당해제한다.
   * 해당 문제는 SharedPtr.h에 구현된다. 이 헤더파일은 “SharedPtr” 클래스가 정의되어 있다. 이 클래스는 “SharedArray”로서 배열에도 적용이 된다.
2. **프로그램의 구조 및 알고리즘**

* 본 프로그램은 포인터와 함께 배열에 대해서도 SharedPtr의 기능을 수행할 수 있도록 using 구문을 통해 SharedArray의 이름으로도 구현한다. 그래서 할당해제 시 기존 포인터와 다른 배열의 할당해제 구조 때문에, 기존 포인터의 할당해제를 수행하는 Deallocator 함수뿐만 아니라 배열의 할당해제를 수행하는 ArrayDeallocator 함수도 구현한다.
* SharedPtr class

SharedPtr class는 템플릿 클래스로서, 템플릿에 두 개의 인수를 받아들인다.

template<typename ObjectType, DeallocatorFuncType<ObjectType> Dealloc = Deallocator<ObjectType>>

ObjectType는 포인터의 자료형을 의미하고, DeallocatorFuncType<ObjectType> Dealloc = Deallocator<ObjectType>>는 할당 형태를 구분짓기 위해 받아들이는 것이다. 기존 포인터인 경우 Dealloc은 Deallocator함수를 의미하게 되고, 배열일 경우 ArrayDeallocator를 의미하게 된다.

* + - private
      * ObjectType\* m\_object : 스마트 포인터의 대상이 되는 포인터를 저장한다.
      * m\_ref\_counter : 해당 메모리를 참조하는 포인터의 개수를 카운팅한다.
    - public
      * SharedPtr() : “ShardPtr<MyClass> ptr;” 형태의 스마트포인터 생성시 작동
      * explicit SharedPtr(ObjectType\* ptr) : “SharedPtr<MyClass> ptr(new MyClass())” 형태의 스마트포인터 생성시 작동.
        + SharedPtr 타입의 ptr에 대해서, ptr = new ~ 이런 형식으로 새로 동적할당된 메모리 주소를 바로 대입할 수 없기에, explicit을 통해서 새로 동적할당된 메모리 주소를 대입할 때 발생하는 자동 형변환을 차단한다.
      * SharedPtr(const SharedPtr& ptr) : “SharedPtr<MyClass> ptr2(ptr);”형태의 스마트포인터 생성시 작동한다. copy constructor의 기능을 수행한다.
      * ~SharedPtr() : destructor. 객체의 동적할당 해제시 작동하며, 만약 해당 객체가 가리키는 메모리를 참조하는 포인터가 하나도 남아있지 않게 된다면, 해당 메모리 또한 할당해제시켜준다.
      * SharedPtr& operator=(const SharedPtr& mtr) : assignment operator이다. 만약에 A=B에서, B가 null pointer가 아니라 어떠한 메모리를 참조하고 있는 포인터라면, 원래 A가 참조하고 있던 메모리를 참조 중단해야 하므로 해당 과정을 수행하는 코드를 넣어주었다.
      * ObjectType\* operator ->() : ->연산자의 기능을 수행하도록 overloading했다. const objectType에 대해서도 기능을 수행할 수 있도록 const버전도 작성했다.
      * ObjectType& operator \*() : \*연산자의 기능을 수행하도록 overloading했다. const objectType에 대해서도 기능을 수행할 수 있도록 const버전도 작성했다.
      * &operator[](int i) : 배열형태의 ObjectType의 경우 배열의 각 원소에 접근하기 위해, [] 연산자 기능을 수행하도록 overloading했다. const 버전도 작성했다.
      * operator ObjectType\*() const : 일반 포인터로 변환하는 typecasting operator을 overloading했다. const 버전도 작성했다.

1. **토론 및 개선**
   * + SharedPtr 템플릿 클래스를 객체지향 프로그래밍이나 제네릭 프로그래밍 관점 또는 다른 측면에서 개선할 수 있다면 어떤 식으로 개선할 수 있을까?

* 몇 개의 포인터가 메모리를 참조하고 있는지 알 수 있게 m\_ref\_counter를 반환하는 함수를 구현하면 스마트 포인터를 사용함에 있어서 매우 편리할 것으로 보인다.
* 해당 메모리에 저장된 포인터가 null 포인터인지 아닌지 판별하는 함수도 구현되면 좋을 것 같다.

1. **참고 문헌**
   * + <https://en.cppreference.com/w/cpp/memory/shared_ptr>
     + https://learn.microsoft.com/ko-kr/cpp/standard-library/shared-ptr-class?view=msvc-170

**2번문제**

1. **프로그램 개요**
   * 본 프로그램은 BMP 이미지파일을 각 픽셀의 RGB값을 분석하여 이에 해당하는 문자로 변환하여 ASCII ART를 만드는 프로그램이다. 이때 각 이미지를 구성하는 모든 픽셀에 대한 정보를 배열에 저장하게 되는데, 이 배열을 1번문제에서 구현한 스마트포인터를 .사용하여 구현한다.
2. **프로그램의 구조 및 알고리즘**

* Image class
  + - private
      * SharedArray<PixelType> m\_buff : rgb 데이터 값을 저장할 배열
      * size\_t m\_width : 이미지의 너비를 저장하는 변수
      * size\_t m\_height : 이미지의 높이를 저장하는 변수
    - public
      * Image() : default constructor
      * Image(size\_t \_width, size\_t \_height) : 이미지의 너비와 높이를 인수로 받아서 그 크기만큼 m\_buff에 메모리를 할당한다.
      * Image(size\_t \_width, size\_t \_height, const PixelType& val) : 이미지 크기만큼 메모리를 할당한 후, val로 모든 배열의 원소를 초기화한다.
      * Image(const Image<PixelType>& img) : copy constructor의 기능을 한다. 이때 deep copy가 되도록, 메모리를 새로 할당한후 원소 하나하나씩 복사한다.
      * ~Image() : destructor
      * Image& operator=(const Image& img) : 대입연산자. 이것도 copy constructor와 같이 deep copy가 되도록 메모리를 새로 할당한 후 원소 하나하나씩 복사한다.
      * size\_t width() const : 이미지의 너비 값을 반환한다.
      * size\_t height() const : 이미지의 높이 값을 반환한다.

1. **토론 및 개선**
   * + 직접 만든 스마트 포인터가 어떻게 적용될 수 있는지 경험해 볼 수 있었다. 이전까지는 클래스 정의를 위한 헤더파일을 작성했지만, 이번에는 그렇게 정의하고 구현한 클래스를 이용해서 직접 유용한 프로그램의 헤더파일로 사용하여 프로그램을 구현하는 것이 마치 “string” 헤더파일과 같은 우리가 자주 사용하는 헤더파일을 직접만든 것 같아 뿌듯했다.
2. **참고 문헌**