CSED 232 Object-Oriented Programing (Spring 2023)

Programming Assignment # 4

- Template & STL -

Due date : 5 월 19 일 23 시 59 분

담당 조교 : 김우혁 (woohyeok@postech.ac.kr)

주의 사항

- 클래스 선언 및 정의를 main.cpp 파일에 작성하는 것을 금지합니다. 그 외에 선언 및 정의의 위치에 대한 제약은 없습니다.
- STL 및 shared_ptr 을 제외한 모든 C++ 문법이 사용 가능합니다.
- 문제에서 제공한 형식을 준수하고 각 문제별 추가적인 세부 조건을 만족하여야 합니다.
- 문제에 명시되어 있지 않더라도 소멸자(Destructor)와 같은 메모리 누수 방지를 위해 필요한 멤버함수는 필수적으로 구현되어야 합니다.
- 문제 조건이 복잡합니다. 모든 문제의 세부 조건을 꼼꼼히 읽어 보시기 바랍니다.

감점

- 제출 기한에서 하루(24 시간) 늦을 때마다 20%씩 감점
  - 1 일(20%) , 2 일(40%), … 5 일(100%)
- 컴파일이 정상적으로 이루어지지 않을 경우 0 점

제출방식

채점은 Windows Visual Studio 2022(윈도우 사용자의 경우) 및 Ubuntu 20.04(lts)와 gcc version 9.4.0 (맥 사용자의 경우) 환경에서 이루어집니다. VS 로 작업했을 경우 작업하신 환경이 있는 visual studio 프로젝트 폴더에 Report 를 포함하여 zip 파일로 압축 후 제출해 주시기 바랍니다. (x64 및 .vs 폴더는 전부 지워주십시오) 마찬가지로, 맥 이용자의 경우 소스 코드, 보고서, Makefile 을 포함한 폴더를 압축해서 제출해주시면 됩니다. 폴더명은 '학번'으로 만들어 주시고, Report 는 docx 나 pdf 형식으로 제출해주세요. 반드시 PLMS 를 통해 제출해주시기 바랍니다. 이메일 제출은 인정되지 않습니다.

공통 채점 기준

1. 프로그램 기능
   - 프로그램이 요구 사항을 모두 만족하면서 올바로 실행되는가?

2. 프로그램 설계 및 구현
   - 요구 사항을 만족하기 위한 변수 및 알고리즘 설계가 잘 되었는가?
   - 문제에서 제시된 세부 조건을 모두 만족하였는가?
   - 설계된 내용이 요구된 언어를 이용하여 적절히 구현되었는가?

3. 프로그램 가독성
   - 프로그램이 읽기 쉽고 이해하기 쉽게 작성되었는가?
   - 변수 명이 무엇을 의미하는지 이해하기 쉬운가?
   - 프로그램의 소스 코드를 이해하기 쉽도록 주석을 잘 붙였는가?

4. 보고서 구성 및 내용, 양식
   - 보고서는 적절한 내용으로 이해하기 쉽고 보기 좋게 잘 작성되었는가?
   - 보고서의 양식을 잘 따랐는가?

다른 사람의 코드나 인터넷에 있는 프로그램을 복사(copy)하거나 간단히 수정해서 제출하면 무조건 'F' 학점이 부여됩니다. 이러한 부정행위가 발견되면 학과에서 정한 기준에 따라 추가적인 불이익이 있을 수 있습니다.

Shared Pointer & Image Library

Objective

본 과제에서는 Standard Template Library 에서 제공하는 기능 중 하나인 shared_ptr 을 직접 구현해 보고 이를 이용한 영상 처리 클래스를 구현하여 봄으로써 template 과 STL 에 대한 이해를 높인다.

문제 1 - SharedPtr

1.1. 개요

본 과제의 1 번 문제는 shared_ptr 의 간소화된 버전인 SharedPtr 템플릿 클래스를 구현해 본다. 스마트 포인터(smart pointer)는 메모리 누수로부터 프로그램을 보호하기 위해 C++의 Standard Template Library (STL)에서 제공하는 기능이다. 스마트 포인터는 포인터처럼 동작하는 클래스 템플릿으로 사용이 끝난 메모리를 자동으로 동적해제해 주는 것이 특징이다. 일반적인 포인터를 사용할 때에는 new 연산자를 통해 실제 메모리를 가리키도록 선언하며 사용이 끝난 포인터에 대해서는 delete 연산자를 통해 메모리를 수동으로 해제해야 한다. 이러한 방식은 메모리 관리의 어려움을 유발하며 메모리 누수 현상이나 메모리 문제로 인한 런타임 에러를 빈번하게 발생시키게 된다. 스마트 포인터는 개발자가 일일이 delete 를 할 필요 없이 메모리의 사용이 끝나면 자동으로 해제하는 방식으로 이러한 어려움을 해결해 준다.

C++의 STL 은 여러 종류의 스마트 포인터를 제공하는데 그 중 하나가 shared_ptr 이다. shared_ptr 은 memory 헤더파일을 include 하면 사용 가능하다. shared_ptr 은 동적 할당된 메모리 영역이 현재 더 이상 사용하지 않는 영역인지를 파악하기 위해 참조 카운트 (reference count) 방식을 사용한다. 참조 카운트를 이용한 스마트 포인터 동작 방법에 대해서는 강의 자료 "16. The string Class and the Standard Template Library (1)"과 본 문서의 부록을 참고하기 바란다.

1.2. 과제 요구사항

본 과제에서 구현해야 하는 SharedPtr 의 기능은 다음과 같다.

- SharedPtr 생성
  - SharedPtr 은 template class 로 적어도 하나 이상의 template parameter 를 받는다. 만약 MyClass 라는 클래스의 객체를 가리키는 포인터를 생성할 경우 다음과 같은 코드를 이용하여 SharedPtr 객체를 생성할 수 있어야 한다.
    - SharedPtr<MyClass> ptr;
  - 또한 아래의 예제 코드와 같이 SharedPtr 객체를 생성할 때 동적 할당된 메모리 영역을 이용하여 SharedPtr 객체를 초기화시킬 수 있다.

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

- SharedPtr 객체를 같은 타입의 객체를 가리키는 다른 SharedPtr 객체를 이용하여 초기화시킬 수 있다.

  - SharedPtr<MyClass> ptr(new MyClass());
    SharedPtr<MyClass> ptr2(ptr);

- SharedPtr 대입 연산

  - SharedPtr 객체를 다른 SharedPtr 객체에 대입할 수 있다. 이 경우 두 SharedPtr 객체는 공통된 동적할당된 메모리를 가리키게 된다.

    - SharedPtr<MyClass> ptr(new MyClass());
      SharedPtr<MyClass> ptr2;
      ptr2 = ptr;

  - SharedPtr 객체에 새로운 동적할당된 메모리 주소를 바로 대입할 수는 없다. 만약 이미 만들어진 SharedPtr 객체에 새롭게 동적할당된 메모리 주소를 대입하기 위해서는 SharedPtr 의 생성자를 이용해 객체를 새로 만들고 대입해야 한다.

    - SharedPtr<MyClass> ptr;
      ptr = new MyClass(); // must raise a compile error!!
      ptr = SharedPtr<MyClass>(ptr); // 이렇게 해야 함.

- SharedPtr 객체의 이용

  - SharedPtr 객체는 자신이 가리키는 객체를 이용할 수 있게 하기 위해 다음과 같은 두 가지 연산자를 지원해야 한다 (*, ->). 두 연산자 모두 const 버전과 non-const 버전을 지원해야 한다.

    - SharedPtr<MyClass> ptr(new MyClass);
      ptr->some_method();
      (*ptr).some_method();
      const SharedPtr<MyClass> const_ptr(new MyClass);
      const_ptr->some_const_method();
      (*const_ptr).some_const_method();

  - SharedPtr 객체는 필요시 일반 포인터로 변환될 수 있어야 한다.

    - SharedPtr<MyClass> ptr(new MyClass);
      MyClass* ptr2 = (MyClass*)ptr;
      const MyClass* ptr3 = (const MyClass*)ptr;

- 자동 동적 해제
  - SharedPtr 객체로 더 이상 참조되지 않는 메모리 영역은 자동으로 동적 해제된다. 아래는 이 동작에 대한 예이다.
    - SharedPtr<MyClass> ptr(new MyClass(1)); // 첫번째 객체 동적할당
      SharedPtr<MyClass> ptr2(ptr);
      ptr = SharedPtr<MyClass>(new MyClass(2)); // 두번째 객체 동적할당. 첫번째 객체는 ptr2 에서 가리키고 있으므로 계속 유지됨
      ptr2 = SharedPtr<MyClass>(new MyClass(3)); // 세번째 객체 동적할당. 첫번째 객체는 더 이상 아무 SharedPtr 객체가 가리키지 않으므로 자동으로 동적해제.
- 동적할당된 배열의 지원
  - 본 과제에서 작성하는 SharedPtr 객체는 배열의 동적할당도 지원한다. 배열의 경우 동적할당을 위해 new 와 delete 대신 new[]와 delete[] 연산자를 사용해야 한다. 이를 위해 SharedPtr 템플릿은 두번째 template parameter 로 deallocation 을 위한 함수를 입력받아서 일반 객체와 배열의 메모리 해제를 다르게 처리하는 것을 지원한다. 이에 대한 구현은 본 과제에 같이 제공된 skeleton code 를 참고하기 바란다.
    - template<typename T> void ArrayDeallocator(T* ptr) { delete[] ptr; }
      SharedArray<int,ArrayDeallocator> ptr(new int[N]); // 두번째 template parameter 로 Array 를 해제하기 위한 함수를 파라미터로 지정할 수 있다. 두번째 template parameter 가 지정되지 않는 경우에는 일반 객체를 해제하기 위한 함수가 default parameter 로 지정된다.
  - 같이 제공된 skeleton code 에서는 배열을 위해 다음과 같이 정의된 SharedArray 라는 타입을 제공한다.
    - Template<typename T>
      using SharedArray = SharedPtr<T,ArrayDeallocator<T> >;
  - 본 과제에서 작성해야 하는 부분은 동적할당된 배열의 지원을 위한 배열 원소 접근 연산자이다. ([ ])
    - SharedArray<MyClass> ptr(new MyClass[10]);
      ptr[0].some_method();
      const SharedArray<MyClass> ptr(new MyClass[10]);
      ptr[2].some_const_method();

- 기타 요구 사항
  - SharedPtr 템플릿 클래스를 객체 지향 프로그래밍이나 제네릭 프로그래밍 관점 또는 다른 측면에서 개선할 수 있다면 어떤 식으로 개선할 수 있을지 보고서에 기술할 것.
  - 첨부된 코드 중 SharedPtr_test.cpp 은 SharedPtr 클래스 템플릿이 제대로 구현되었는지 학생이 스스로 확인할 수 있는 간단한 코드들이 작성되어 있다. 그러나 본 코드는 참고용으로 제공된 것일 뿐 본인의 클래스가 여러가지 상황을 고려하여 잘 구현되었는지 여부는 직접 확인해 볼 것을 추천한다. 조교는 해당 sharedPtr_test.cpp 가 아닌 더 복잡한 상황을 테스트해 볼 것이다. 참고로 SharedPtr 클래스 템플릿이 제대로 구현되었다면 SharedPtr_test.cpp 컴파일 후 결과는 다음과 같아야 한다.

```
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
```

문제 2 - Image

2.1. 개요

본 과제의 2 번 문제는 이미지 처리를 위한 기초적인 템플릿 클래스 작성이다. 이미지는 컴퓨터에서 일반적으로 픽셀들의 2 차원 배열로 표현되며, 각각의 픽셀에는 밝기 정보나 색상에 대한 정보가 저장된다. 컬러 이미지의 경우에는 각 픽셀에 Red, Green, Blue 의 세가지 색상에 대한 정보가 저장되며 그레이스케일 (grayscale) 이미지의 경우에는 각 픽셀에 밝기 정보가 저장된다. 일반적으로 색상 정보나 밝기 정보는 0 부터 255 사이의 값을 갖는 8 bit unsigned integer 타입을 이용하여 저장된다. 따라서 그레이스케일 이미지는 픽셀마다 8 비트를 사용하며 컬러 이미지는 픽셀의 각 색상마다 8 비트를 사용하여 총 24 비트를 사용한다. 이미지의 픽셀마다 수치 연산을 적용하는 경우에는 8 비트 정수형 타입은 각종 수치 연산에 적합하지 않기 때문에 float 이나 double 과 같은 부동소수점 실수형 타입을 이용하기도 한다.

본 과제에서는 이러한 이미지를 다루기 위한 템플릿 클래스를 작성한다. 이를 통해 이미지 파일 포맷 중 하나인 BMP 파일로부터 이미지를 읽어서 간단한 영상 처리를 거친 후 나온 결과를 BMP 파일로 저장할 수 있는 프로그램을 작성해 본다. 또한 BMP 파일로 읽은 그림을 문자로 변환하여 출력하는 코드도 작성한다.

2.2. 과제 요구사항

본 과제에서 구현해야 하는 Image 클래스의 기능은 다음과 같다.

- Image 생성
  - Image 클래스 템플릿은 다양한 픽셀 타입을 지원한다. 이를 위해 픽셀 타입을 템플릿 파라미터로 받는다. 예제 코드는 다음과 같다.
    - Image<uint8_t> img; // 픽셀값으로 8 비트 unsigned integer 타입을 사용하는 그레이스케일 이미지 객체 생성. uint8_t 는 8 비트 unsigned integer 타입으로 cstdint 에 선언되어 있음.
      Image<float> imgf; // 픽셀값으로 float 타입을 사용하는 그레이스케일 이미지 객체 생성
      Image<RGB<uint8_t>> rgbimg; // 픽셀값으로 8 비트 정수형을 사용하는 RGB image 생성. RGB<> 클래스 템플릿은 제공된 skeleton code 참고
- Image 클래스 템플릿에서 제공하는 public interface
  - 생성자
    - Image() // default constructor

- ◆ Image(size_t _width, size_t _height); // 이미지 객체 생성 시 이미지의 크기만큼 메모리 할당 (_width x _height)
- ◆ Image(size_t _width, size_t _height, const PixelType& val); // 이미지 객체 생성 시 이미지의 크기만큼 메모리 할당하고 val 값을 이용해 모든 픽셀값 초기화
- ◆ Image(const Image<PixelType>& img); // copy constructor

■ 소멸자

- ◆ ~Image(); // 특별히 아무 일도 하지 않음.

■ 연산자

- ◆ 대입 연산자 (operator=)– 같은 픽셀 타입을 사용하는 이미지 객체로부터 대입받는연산자. 다음과 같은 동작을 수행할 수 있어야 함.
  - const Image<float> a;
    Image<float> b;
    b = a;
- ◆ 배열 액세스 연산자
  - 이는 따로 구현할 필요 없음. skeleton 코드에 이미 구현되어 있으니 참고바람

■ 기타 멤버 함수

- ◆ size_t width() const; // 이미지의 가로 길이 리턴
- ◆ size_t height() const; // 이미지의 세로 길이 리턴

● 기타 요구 사항

■ Image 클래스의 픽셀 값을 저장하기 위한 공간은 동적 할당을 이용해 구현해야 한다. 이 때 문제 1 에서 구현한 SharedArray 를 이용하여 구현할 것.

■ Image 템플릿 클래스를 객체 지향 프로그래밍이나 제네릭 프로그래밍 관점 또는 다른 측면에서 개선할 수 있다면 어떤 식으로 개선할 수 있을지 보고서에 기술할 것.

■ 본 과제에 같이 제공된 image_test.cpp 는 Image 클래스 템플릿을 이용한 간단한 영상 처리 예제 코드로 Image 클래스 템플릿이 잘 구현되었다면 다음과 같은 내용을 출력하게 된다. 이 외에도 본인이 구현하고 싶은 다른 영상 처리 예제나 추가 기능이 있다면 구현하고 이를 보고서에 기술할 것. Image 클래스 역시 조교는 image_test.cpp 에 제시된 코드 외에도 더 복잡한 상황을 테스트해 볼 것이다.

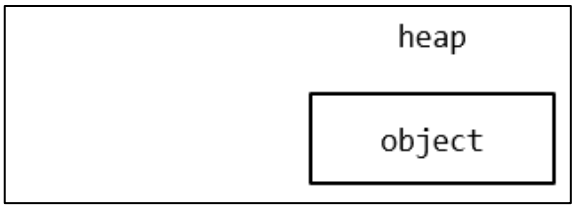
```
    Dealloc Array
    Dealloc Array
    Dealloc Array
    Dealloc Array
    Dealloc Array
    2. ASCII conversion
    2.1. Grayscale conversion
    Dealloc Array
    2.2. Downsampling
    Dealloc Array
    Dealloc Array
    Dealloc Array
    2.3. ASCII art drawing
    OOOOOOOOOOOOOOOOOOOOOOOOpEo3Cii}t5SVOOOOOOOOOOOO
    OOOOOOOOOOOOOOOOOOOOOOde())JJJJJJJ7(uSpOOOOOOOOOO
    OOOOOOOOOOOOOOOOOOOOOx7)))))JJJJJJJ777iZhOOOOOOOO
    OOOOOOOOOOOOOOOOOOptv)))))))JJJJJJJ77777Ij6pOd]p
    OOOOOOOOOOOOOOOOOOp3vvv)))))))JJJJJJJ77J7777(((CO
    OOOOOOOOOOOOOOOOOOevvv)v)))))))JJJJJJJ7777777(ZO
    OOOOOOOOOOOOOOOOOkvvvvvv)))))))))JJJJJJJJ77777|9O
    OOOOOOOOOOOOOOOOOfTvvvvvv)v)))))))JJJJJJJJ7777yOO
    OwOOOOOOOOOOOOOOO6TTTvvvvvvv)))))))JJJJJJJJ77YOOO
    OFfwOOOOOOOOOOOp5TTTvvvvvvv))v)))))))JJJJJJIqVdOO
    OZ(7[4OOOOOOOOOOVLTTTTvvvvvvT+v)))))))Jfnj6dyidOO
    OVI77F2OOOOOOOOOOT=zTTTTvvT!^Lv)))))))}Z[3}FJJfOOO
    Op3JJJ)19OOOOOOOOSk1vTTTTzCjpy)vv)))))7))JJJJJEOOO
    Olv7JJ))J5pOOOOOOSz^5TTTT3E/vV[)v)))))))))JJJeOOOO
    Ov)JJ))vvTFSOOOOOu``[)JTTo/.`xjvvvvv)))))))|yOOOOO
    Oy()))vvTTTL19OOd97vtfff7}I`=htvvvvv))))Jt2pOOOOOO
    OOpP))vvTLLLsv5pqFo3fffffTuShovvvvvvv))eSaZ3dOOOOO
    OOOOe(vTTLLsLLL|ZLvifFF7FJTTTTTTvvvvvv}|)))ZOOOOOO
    OOOOpSSFLLssLLLLsLLL|+`.FTTTTTTTTvvvvvvv))fpOOOOOO
    OOOOOOOpYJLLLLLLLLsLTFJi7LTTTTTTTvvvvvvvvf4OOOOOOO
    OOOOOOOOOVZ)LLLLLLLsLLTLLLTTTTTTTTTTvvvvnpOOOOOOOO
    OOOOOOOOOOVx|LLLLLLLLLLLLLLTTTTTTTTvv}wOOOOOOOOOO
    OOOOOOOOOOOOOwl)sLLJLs))LLLTTTTTTT(uwOOOOOOOOOOOO
    OOOOOOOOOOOOOOOpFsl{exfCfLLLLLsJo9OOOOOOOOOOOOOOO
    OOOOOOOOOOOOOOOOI)CC{LtJ3LLLLLLLsiqOOOOOOOOOOOOOO
    OOOOOOOOOOOOOOOO[LeCLFLl3LLLLLsLLLL5OOOOOOOOOOOOO
    OOOOOOOOOOOOOOOOYsYTJ5L}1LsLLL7sLsLL[OOOOOOOOOOOO
    OOOOOOOOOOOOOOOOETIlLs7f3LLLLLPIsLTTvYOOOOOOOOOOO
    OOOOOOOOOOOOOOOOO9TITlIfs1LLsss9p3LTvvJkOOOOOOOOO
    OOOOOOOOOOOOOOOOOv{I3f337LLLLLdOpIvv)JuOOOOOOOOOO
    OOOOOOOOOOOOOOOOO{ssLsLLLssLLs4OOV{)J(EOOOOOOOOOO
    OOOOOOOOOOOOOOOOOuLsLsLLLLLLLsVOOO6(C3EOOOOOOOOOO
    OOOOOOOOOOOOOOOOOjLLLLLLLLLLLspOOOO]o5OOOOOOOOOOO
    OOOOOOOOOOOOOOOOOPsLLLLLLLLLLLpOOOOO4OOOOOOOOOOOO
    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
```

부록 – 참조 카운팅 (Reference counting)

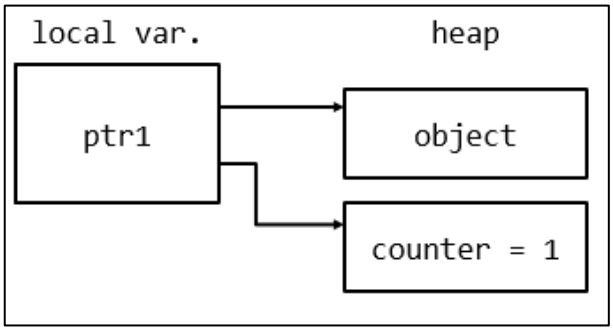
STL 의 shared_ptr 은 참조 카운팅을 이용하여 동적 할당된 객체를 관리한다. 동작 원리는 다음과 같다. 아래와 같은 코드가 있다고 하자.

```
1: {
2:   string* tmp = new string("Some String");
3:   shared_ptr<string> ptr1(tmp);
4:   {
5:     shared_ptr<string> ptr2;
6:     ptr2 = ptr1;
7:   }
8: }
```

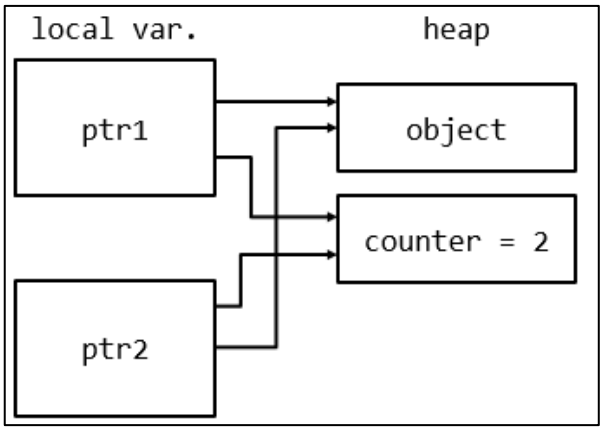
위의 코드에서 2 번 라인까지 실행되었다고 하자. 이 때 tmp 가 가리키고 있는 string 객체는 heap 영역에 동적 할당된다. 이때 heap 영역을 다음과 같이 나타낼 수 있다.



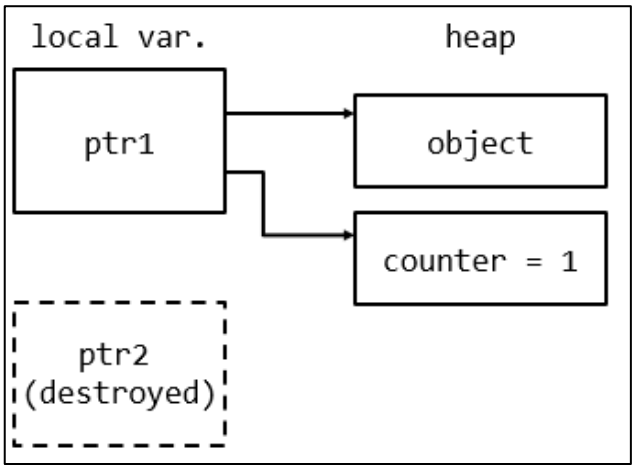
3 번 라인이 실행되면서 shared_ptr 의 객체인 ptr1 이 생성되고 ptr1 은 tmp 가 가리키던 string 객체를 가리키게 된다. 이 때 마찬가지로 heap 영역에 이 object 를 몇 개의 shared_ptr 객체가 가리키고 있는지를 카운팅하는 변수 (counter)를 생성한다. 그리고 현재 ptr1 이 object 를 가리키고 있으므로 카운팅 변수를 1 로 초기화한다. 이 상태를 도식화하면 다음과 같다.



5 번 라인이 실행되면서 shared_ptr 객체인 ptr2 가 생성된다. 그리고 6 번 라인이 실행되면서 ptr2 에 ptr1 의 값을 대입한다. 이 대입연산에서 ptr2 는 ptr1 이 가리키고 있던 object 를 같이 가리키면서 동시에 counter 도 가리키게 된다. 또한 이 때 counter 의 값을 1 증가시킨다.



7 번 라인이 실행되면서 안쪽 블록을 벗어나게 되고 블록 내의 local variable 인 ptr2 객체가 소멸된다. ptr2 객체가 소멸되면서 counter 를 1 감소시킨다.



마지막으로 8 번 라인이 실행되면서 바깥쪽 블록을 벗어나게 되고 블록 내의 local variable 인 ptr1 객체가 소멸된다. ptr1 객체가 소멸되면서 다시 counter 를 1 감소시킨다. 그리고 감소된 counter 가 0 이 되었다면 ptr1 객체의 소멸자는 counter 와 object 를 모두 동적 해제한다.