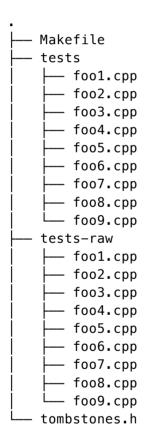
# **A5 Tombstones**

- Division Labor:
  - o Chi Chun Chen
  - Shaojie Wang
  - We do most of the parts together in this assignment.

### File Structure

- Testing programs under tests use tombstone to check memory leak and dangling references.
- Testing program under tests-raw has the check turned off.



#### How to run

• Compile all the tests in test/

make

• Compile all the tests without tombstone

https://hackmd.io/s/S1DjJ1RkM

make raw

- After the executing of the makefile, the executables lists in the current directory, so just type ./foo<n> to execute it, which n can be number 1~9.
- The tests given by assignment webpage might have some invalid actions that will terminate the program if we use raw pointer instead of Pointer<T>
  - o foo2.cpp
    - Double free an object
  - foo7.cpp
    - Dereference NULL pointer

#### **Extra Credit**

- Use a non-type parameter in the template of Pointer class so that we can turn-off the checking at compile time.
  - Syntax for turn on tombstone
    - Set the first template parameter to true
    - Pointer<int, true> foo(new int(12));
  - Syntax for turn off tombstone
    - Switch the second template parameter to false
    - Pointer<int, false> foo(new int(12));
- Inheritance works correctly, the test case for inheritance is inside test/foo9.cpp and test-raw/foo9.cpp
- Extra credit #3
  - Should T\* and Pointer<T> be interoperable?
    - In most cases, yes. See the following explanations.
  - Should you be able to use one in a context that expects the other?
    - If we want to cast Pointer<T> into T\*, then yes, because the casting simply returns a T\* raw pointer.
    - If we want to cast T\* to Pointer<T> on the fly, such as in cout statement, then no, because this may cause a memory leak due to no assignment.
  - Should you be able to assign one into the other? If not, explain why. If so, implement the necessary support routines.
    - It is possible to assign one into the other
      - We implement it using conversion constructor and put the tests in tests/foo10.cpp and tests-raw/foo10.cpp
    - Also, the conversion has no affect on reference count since we create one Pointer object in the assignment, which happens after the conversion constructor
  - If Pointer<T> interoperates with T\*, how about T&?
    - First, a reference cannot be assigned to be NULL, and our Pointer<T> can be constructed using NULL as given parameter

https://hackmd.io/s/S1DjJ1RkM

 Second, Pointer<T> is able to be re-assigned if freeing it before assignment (so that no memory leak happens)

#### **Basic**

- Use tombstone and reference count to detect possible dangling references and memory leaks.
  - template <class T> struct Tomb has T\* content and int ref\_cnt as properties,
    and Pointer class has a Tomb\* variable as protected property.
  - Reference count changes at:
    - Default constructor: ref\_cnt is 0 and content is NULL
    - Copy constructor: if content is NULL , then ref\_cnt is 0. Otherwise, ref\_cnt++ .
    - Boostrap constructor: if content is NULL, then ref\_cnt is 0, otherwise 1.
    - Assignment: original ref\_cnt--, then the same as copy constructor.
    - free(Pointer<T>&) : set ref\_cnt 0 and content NULL .
    - Destructor: ref\_cnt--.
  - Dangling references are checked when deferencing and freeing the Pointer object.
    If operator \* deferences a Pointer with NULL content, or a Pointer object is freed with more than one reference count, then we raise a dangling reference error.
  - Memory leaks are checked in the overloading of = operator and the destructor. If the reference count of a tombstone goes to zero while the content is not NULL, then we raise a memory leak error.

## **Run Time Error Message**

- Dangling Reference
  - The line number we print helps us knowing in which place did tombstones.h triggers the dangling reference.

Dangling reference at tombstones.h line: 279

- Memory Leak
  - Same idea is used for printing memory leak message.

Memory leak at tombstones.h line: 150

https://hackmd.io/s/S1DjJ1RkM