RAII & Smart Pointers & Representation Invariants & Abstraction Functions

CS 247

University of Waterloo
cs247@uwaterloo.ca

June 2, 2017
Overview

1. RAII
   - Three levels of Guarantee
   - What’s RAII

2. Smart Pointers

3. Representation Invariants

4. Abstraction Functions

5. More Examples
   - Bubble Tea
   - Company
Three levels of Guarantee

- Basic guarantee: if an exception is thrown, data will be in a valid state but may not make sense.
- Strong guarantee: if an exception is thrown, the data will appear as if nothing happened.
- No-throw guarantee: an exception is never thrown.
RAII: Resource Acquisition is Initialization
RAII is vital to writing exception-safe code in C++. Mechanism: when an exception is thrown, stack-allocated memory will be reclaimed
unique_ptr is similar to auto_ptr but supports more functionality, only one unique_ptr can point to the same block of heap memory.

shared_ptr allows many pointers that all point to the same block of heap memory and only deletes that memory when no other shared_ptrs point to it.

Example: smartptr.cc
A Representation Invariant is a property of an ADT that must be true at all times for the ADT’s state to be considered valid. Formally, a representation invariant is a boolean expression - something that evaluates to “true” or “false”.
For example, in a max-heap: for every node $N$, $N$’s child nodes must have keys that are smaller than $N$. (the largest key is in the root node.)
Induction

Typically, you can use inductive reasoning to prove that your Representation Invariant will hold.

For heap order property:

- Base Case:
- Inductive Hypothesis:
- Inductive Step:
A Representation Invariant is an assumption that you make about the code that you’re writing - but sometimes, your assumptions can be wrong. Representation Invariants should give you a good idea of what you should be asserting in your code.
In the code for your heap, you can write a function `bool checkHeapProperty()` to verify if a heap is valid.

class Heap {
    Heap(); // check here
    void insert(item); // check here (before and after)
    void deleteMax(); // check here (before and after)
    item getMax();
};
An Abstraction Function is a mapping of the code implementation that implements an ADT to the abstract model of that same ADT that we have in our heads.
Heap Abstraction Function

Representing a heap with $k$ elements using an array of size $k$:

$AF(r) =$ a heap such that for all nodes $n$ in heap,

8,7,5,4,3,4,1,2,3,2
Bubble Tea is a tea-based drink originating from Taiwan. It consists of:
- a base (green or black tea),
- milk (or no milk),
- a flavour (e.g. strawberry, blueberry, taro, kiwi, chocolate, coconut...)
- extra toppings (e.g. tapioca balls, various flavours of jelly)
In code, we can represent a Bubble Tea object like so:

class BubbleTea {
    int base_; // 0 = green, 1 = black
    bool hasMilk_;
    string flavour_;  
    string[] extraToppings_;
};
Representation Invariant:
What do you expect must always be true of a serving of Bubble Tea for it to be valid (i.e. for them to be able to make it and sell it to you?)

Abstraction Function: How does the code version of a Bubble Tea ADT map to an actual serving of Bubble Tea?
Bubble Tea Example Solution

Representation Invariant:

base_ must be 0 or 1,
flavour_ must come from a list of valid flavours (the menu)
extraToppings_ must also come from the menu, and
extraToppings_.size <= MAX_TOPPINGS

Abstraction Function:

AF(r) =
  Base = B(base_) where B(0) = green, and B(1) = black.
  Has Milk if hasMilk_ is true,
  Flavour = flavour_, and,
  Toppings = all elements in extraToppings_.[].
Consider some representation of a Company:

class Company{
    class Employee{
        string name;
        int num_subord;
        Employee* subord[];
    };
    Employee owner;
};

What would the abstraction function be for mapping this Company ADT to a stafflist of some company. (hint: think recursively)
Abstraction Function:

\[ AF(r) = \{ r.\text{name} \} \cup \{ \} \cup AF(*(r.\text{subord}[0])) \cup \ldots \cup AF(*(r->\text{subord}[\text{num_subord}-1])) \]

A full stafflist of the company corresponds to AF(owner).
The End