CS 246 Fall 2021 — Tutorial 7

October 27, 2021

Summary

1 Nested Classes

- We may want to create a class that doesn't make sense to exist on its own. An example is implementing a "wrapper" class for some structure to restrict others' ability to alter the class.
- A good example of this would be creating a wrapper class around the `Node` class that we implemented.

```
class LinkedList {
    struct Node {
        int val;
        Node* next;
    };

    int numNodes;

    Node *head;
    Node *tail;

public:
    LinkedList();
    LinkedList(int amount, int what); // fill constructor

    void insertHead(int value);
    void insertTail(int value);

    void remove(int index);
};
```

2 Iterators

- Iterators are used to traverse containers in some order.
- The order can be specified by the definition of the iterator, or not specified to be in any order.
- In C++, iterators are usually implemented with the following functions with in the container class (`Container`) and the nested iterator class (`Container::Iterator`):
- These operations are crucial to support the *range-based for loop* for objects:

```
// Assume that Container iterator iterates through objects of type T
Container c;
for (T &v : c) {
    // You can change v here since declared to be of type (T&).
    v.modify();
    cout << v << endl;
}
```

- See the lecture material for the implementation of a linked-list iterator.

2.1 Iterator Examples

There are three different examples of iterator implementations in the provided sample code.

1. `vec.cc` shows how `std::vector` has its own version of an iterator and we can use a ranged for loop to traverse it.
2. `array.cc` shows the creation and use of an iterator that traverses a C-style array of integers in reverse order.
3. `bst.cc` shows how `std::stack` is used by the iterator to remember where in the binary search tree the iterator is in its traversal.

3 Class Relationships

- There are three types of relationships between classes that we typically discuss:
  - **Composition(owns-a):** class A *owns an* instance of class B. This means that class A is responsible for deleting the instance of class B when an object of class A is destroyed.
    - Under composition, instances of B cannot be shared once bound to an A.
  - **Aggregation(has-a):** class A *has an* instance of class B. This means that class A is not responsible for deleting the instance of class B.
    - If an object of class A is deleted, the instance of class B associated with it lives on.

        * Multiple objects of class A can share the same instance of class B.

    - **Inheritance (is-a):** class B *is a* class A. This means that an instance of class B can be used in any situation where an instance of class A can be used.

      Note:

        * The converse is not true. That is, an instance of class A cannot always be used where an instance of class B can be used.
        * In this course, we are mainly concerned with public inheritances.

- **Note:** If a class A has a pointer to an instance of class B, you cannot know if the relationship is composition or aggregation without looking at documentation or code.

```
class B {
    ...
};

class A {
    B b1; // This is composition
    B *b2; // This could be composition or aggregation
};
```

4 Inheritance

- Example:

```
class A {
    int a;

public:
    explicit A(int a) : a{a} {}
};

class B : public A {
    int b;

public:
    B(int a, int b) : A{a}, b{b} {}
};
```

5 Encapsulation and Inheritance

- If A has fields that are private, B cannot access these fields (as they are private within A).
- What are some benefits of an inherited class not having direct access to the fields of the superclass?
- However, we often want to give subclasses "special access" to the class.
- For this purpose, we can use the third type of accessibility: `protected`.
- Members that are `protected` can be accessed directly by subclasses but cannot be accessed by the public.
- **Note:** Most of the time you should not make fields `protected` as this also breaks encapsulation.