CS280 Assignment - Lariat

Luke Cardell, slightly modified by Dmitri Volper

January 30, 2025

1 What is Lariat

> It is important to understand that this is not the only way to solve this assignment, merely the way that worked best for me.

Under the hood, Lariat looks like a "linked list of arrays".

The main goal is to have a more efficient (compared to array) insert in the middle. Reminder: if array (and consequently `std::vector`) insert an element in the middle, all elements to the right of the insert position have to be shifted 1 position to the right to free a spot for new element. This is $O(N)$ (linear) operation. Another possible problem is if there is not anough space, which will required reallocation if the whole array. Lariat solves the problem by storing small segments of contiguous memory non-contiguously.

During insert one of 2 things can happen (see datail later):

- the contiguous block may have extra space, so only some elements have to be shifted (1)
- block containing insert position is full, then we split it (2)

```
//    0  1           2  3  4       // index
// +------------+  +------------+
// |  1  3    -|->|  4  5  6  -|->  // data
// +------------+  +------------+
// after insert 2 before 3
//    0  1  2        3  4  5        6  7  8
// +------------+  +------------+  +------------+
// |  1  2  3  -|->|  4  5  6  -|->|  9  9  9  -|->
// +------------+  +------------+  +------------+
// only value 3 had to shifted
```

Figure 1: Block containing insert position has extra space

```
// insert 8 before 5
// first split:
//    0  1  2        3  4           5  6           6  7  8
// +------------+  +------------+  +------------+  +------------+
// |  1  2  3  -|->|  4  8    -|->|  5  6    -|->|  9  9  9  -|->
// +------------+  +------------+  +------------+  +------------+
// now insert
//    0  1  2        3  4           5  6           7  8  9
// +------------+  +------------+  +------------+  +------------+
// |  1  2  3  -|->|  4  8    -|->|  5  6    -|->|  9  9  9  -|->
// +------------+  +------------+  +------------+  +------------+
// only values 5 and 6 were copied
```

Figure 2: Block containing the insert position is full

2 Luke's Guide to Templated Templates

> "We put some templates in your templates so you could generate code at compile time while you generate code at compile time." -C++

A template is a section of code like any other that gets "written" at compile time, rather than when you actually write templated code. In this sense, writing a template is about using the compiler to your own advantage to avoid doing unnecessary work.

> Laziness FTW

A normal function template is declard as such:

```
template<typename T>
void Foo(T bar);
```

The definition requires knowledge of the template parameters in order to work:

```
template<typename T>
void Foo<T>(T bar) {
        //     |
        //      `-[function template ]

        //...
}
```

In the Lariat assignment, you will have to make a templated copy constructor and assignment operator for the Lariat template class. This is similar to writing a static function template, but it's written within another encompassing template, the Lariat class in this case.

The basic declaration of the function is just like a normal template function, but within a class scope.

```
template<typename T>
class Foo {
        //...

        template<typename Bar>
        void Baz(Bar & qux);

        //...
};
```

The definition of the function takes a little bit more work, which may be confusing at first.

```
template<typename T>    // Gives the external class template information
template<typename Bar>  // Gives the internal function template information
            void Foo<T>::Baz<Bar>(Bar & qux) {
  // |          |
  // |           `-[function signature]
  //  `-[class namespace]

  // ...
}
```

The first template declaration belongs to the class, which may be instantiated multiple times with multiple name-mangled signatures based on the compiler's implementation of templates.

The second template declaration belongs to the function, which will likely be instantiated multiple times for every single class instantiation. In this way the structure resembles a tree:

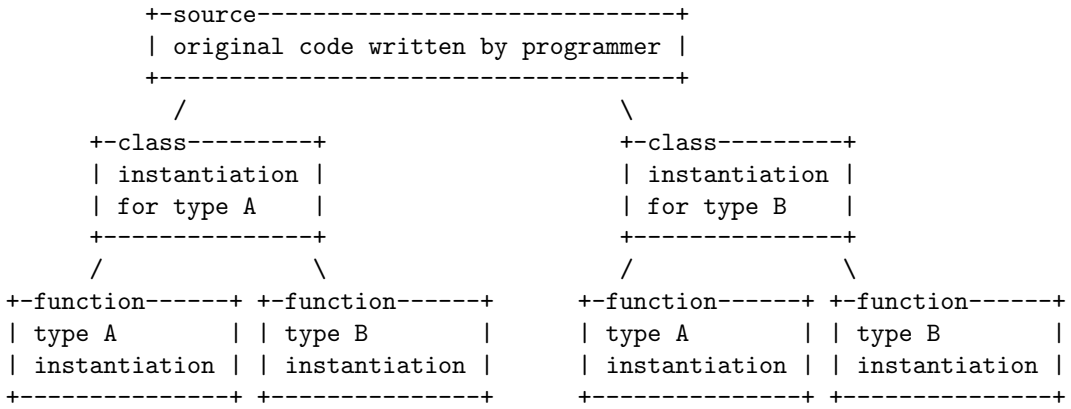

Figure 3: #dat_ascii

The next little template trick you might run into with this assignment comes when you try to refer to internal member types in the instantiation of code. I personally used this in my helper functions that returned pointers to nodes in the list, but it's useful for any further template work you might be interested in doing.

Dependent types are also instantiated for ech class, with the exact same process as function templates. This means that using just the internal name in external code will lead to type confusion, where the compiler isn't sure which owning class is being referred to.

A class class template with a member type and a member function returning a pointer to the member type:

```
template<typename T>
class Foo {
  //...

  struct Bar {
    //...
  }

  Bar * Baz();

  //...
};
```

The member function here must be defined as such:

```
  template<typename T>
              typename Foo<T>::Bar * Foo<T>::Baz() {
    // |       |    |       |     |
    // |       |    |       |      `-[function signature]
    // |       |    |        `-[class namespace]
    // |       |     `-[member typename]
    // |        `-[class namespace]
    //  `-[declaration of token as user-defined type]

    //...
  }
```

The last real trick I want to share with you pertains directly to the copy constructors this assignment requires. At their instantiation, class templates only have direct access to their own instatiation's member variable. This means that in order to see another instantiation's private variables, there must be a declaration of each instantiation as a friend.

> How do you make something to account for any scenario at compile time?
>
> Simple. More templates.
>
> Declare a friend class template in your original class template.

```
template<typename T>
class Foo {
  //...

  template<typename U>  //[nested template declaration]
  friend class Foo<U>;  //[friend declaration for class template]

  //...
}
```

This has been my brief introduction to templates as they pertain to this assignment. they may seem scary, but templates allow for a greater depth and elegance of code, decreasing the total writing time and reducing the volume of duplicated code.