CAAM 420 Scribe Notes

Alex Balkum

Oct 30, 2013

1 C Linkage

Sometimes, you have C code that needs to be used with C++ code. However, as C++ and C have different naming conventions, linkage of the code would be impossible if they were both compiled with a C++ compiler. Thus, if you want to use code compiled in C (rather than C++) use `extern "C"`.

Likewise, if you need to couple C++ and Fortran, use same idea (`extern "C"`).

An example can be seen in the file `clapack.h`. At the beginning of the file:

```
#ifdef __cplusplus
extern "C" {
#endif
```

The preprocessor statement says that if C++ is defined, the following have C linkage.

At the end of the file is the following:

```
#ifdef __cplusplus
}
#endif
```

This ends `extern "C"`.

Some notes on `clapack.h`: These functions allow you to do calculations over complex numbers. Why are there four copies of everything? For both complex and real numbers, there is 1 version for float and 1 for double.

2 Operator Overloading

Why overload operators? Take the assignment operator. Typically, it copies bitwise. However, this should only happen due to your intention. Fact of C++: unless you've overloaded the assignment operator, it will just copy byte by byte. This can lead to memory leaks. For example, if a pointer to memory is reassigned to point to something else. An example of an overloaded assignment operator can be seen in `vecppdense.hh`.

2.1 Operators in vecppdense.hh

Along with the overloaded assignment operator, there are operators for `+` and `*` as well. Why? The purpose of these operators is to supply functionality of MATLAB. For example, assignment now has assigns just as in MATLAB, `+` adds vectors, `*` multiplies a vector and scalar. Thus, you can fake matlab in C++.

3 Strings

3.1 String Basics

- String is a class in the C++ standard library. The type name is obvious: `string`. You might see it prefixed by `std::string`. For future reference, `std::` is a namespace. More on namespaces later.

- C++ lets you use strings in an easy way (rather than char arrays ended by null character).

  Example: `ex60_strings.cc` shows useful things you can do with string class.

3.2 String Applications

- First: you can initialize one with `string stringname`. (Safe to disregard prefix if tell compiler you are using the standard namespace (using namesspace std;). The C++ compiler includes it by default.

- How to see string? Just like double, int, etc. Display using `cout<<stringname<<endl;`. Just like other operators, `<<` is overloaded for the string class to prop-

erly interact. FYI: `endl` is a dumb trick. Can just use `\n`. `endl` is a struct that just puts a newline in a string.

- Cool Thing: you can add strings! To concatenate strings, just add them.
  For example:

  ```
  string f = "fruitcake";
  string f = f + " cook";
  ```

  The variable f will now be equal to `fruitcake cook`. Similar to other types, you could also say `f += " cook"`. The reason why string addition works is because the "+" operator is overloaded. Under their hood, strings are container classes. They are treated like arrays (as they have indexing).

3.3 How to get between C strings and C++ strings

The solution: the `c_str()` function. The function extracts a C string from C++ string. The return value is `const char*`. The string class is a wrapper around a character array. There is a `concat` function that performs same as `+`. However, you have to have allocated enough memory. The beauty of C++ is that the string class takes care of that for you.

4 Templates and Polymorphism

There are two types of Abstraction in C++.

- **Classes** - implements Object Oriented Programming
- **Templates** - implements Generic Programming

These methods gives us two different ways of writing general code. Both are implemented in C++. Thus, we can use both at the same time. This is the strength of C++ over standard Java. Both are ways of implementing Polymorphism.

A polymorphic type is a type whose operations can be applied to values of different types. Classes use runtime polymorphism while Templates

use compile-time polymorphism. In a way, compile-time polymorphism is superior (you get to find errors earlier).

4.1 Template Basics

What's a Template? Templates allow you to do anything that makes sense for a specific type. For example, see ex61_basictemplate.cc.

```
template<typename T>
T addme(T a, T b) {
return a+b;
}
```

In example above, T stands for "thing". Will replace type in for T. The Template will not work if the operation given doesn't make sense. For example, adding char arrays doesn't make sense. However, adding strings will make sense.

The chapter on templates in Stroustrup (Ch. 13) is worth reading (at least first couple sections).