Software Design (C++)
6. Other Useful Things in C++

Antti-Pekka Tuovinen
University of Helsinki

Outline

- String streams
- Function objects & lambdas
- Variadic templates
- Concurrency (Threads, Atomics, Futures)
String streams

- C++ supports reading and writing in-memory streams based on string buffers
  - Can use the same operators and functions as for other streams
- Let’s look at an example in code:

Function objects

- A particularly useful kind of template is the function object (a.k.a functor)
  - used to define objects that can be called

```cpp
template<typename T>
class Less_than {
private:
  const T val; // value to compare against
public:
  Less_than(const T& v) : val(v) { }
  bool operator()(const T& x) const { return x<val; }
};
```

```call operator```
Using functors

- **Functor objects** defined:

  ```cpp
  Less_than<int> lti {42}; // lti(i) will compare i to 42 using <
  Less_than<string> lts {"Backus"}; // lts(s) will compare s to "Backus" using <
  ```

- Can be *called* just like functions:

  ```cpp
  void fct(int n, const string & s)
  {
    bool b1 = lti(n); // true if n<42
    bool b2 = lts(s); // true if s<"Backus"
    // ...
  }
  ```

Using functors

- Function objects are widely used as arguments to *algorithms* (e.g. in STL):

  ```cpp
  void f(const vector<int>& vec, const list<string>& lst, int x, const string& s)
  {
    cout << "number of values less than " << x << " : 
    << count_if(vec.begin(),vec.end(),Less_than<int>{x}) << \\
    << count_if(lst.begin(),lst.end(),Less_than<string>{s}) << "n; \n  }
  ```

  **STL algorithm** - defines the type of the function used as a predicate
  
  **Functor** (predicate)
  
  **Comparison parameter** is *inlined* with the created function object - no need for a global variable or separate function definitions for diff. params!
...but lambdas are even more fun!

- **Lambda** = an anonymous (nameless) function defined on the spot where one is needed:

```cpp
void f(const Vector<int>& vec, const list<string>& lst, int x, const string& s) {
    ...
    cout << count_if(vec.begin(), vec.end(),
                     [&](int a) { return a < x; });
    ...
}
```

Lambdas

- Capture list of *local names* used in the body
  - [ ] capture nothing
  - [=] capture all by value
  - [&] capture all by reference
  - [&x] capture name x by reference
  - [&x,y] capture x by reference and y by value
- Compiler deduces the *return type* automatically from the lambda expression (can also be specified explicitly)
- Usually *short* and *simple inline functions* (1-liners)
Variadic templates

- Since C++11
- A variadic template can take an arbitrary number of template arguments
  - Can be used to create functions that take variable number of arguments
  - Often the better choice because they do not impose restrictions on the types of the arguments, do not perform integral and floating-point promotions, and are type safe
- For an example, see http://en.cppreference.com/w/cpp/language/parameter_pack

Concurrency

- C++ 11 adds
  - Native support for threads
  - A memory model for shared variables
  - Asynchronous function calls (Futures)
- See Hans Boehm’s talk at GoingNative 2012 introducing these concepts:
Concurrency

• C++14 is going to add even more features to support developers who need to write responsive applications
  – See Herb Sutter’s talk at C++ and Beyond 2012 that discusses the current and planned features of C++
  – This is an advanced level talk and it requires at least general understanding of the concepts presented in
    Boehm’s talk