CSE 303
Lecture 25

Loose Ends and Random Topics

slides created by Marty Stepp

http://www.cs.washington.edu/303/

Enumerations (enum)

- **enum**: A type that consists entirely of a set of constant values.
  - Useful for creating a simple constrained type with only a few values
  - in C/C++, an `enum` silently converts to/from `int` type (0-based)

```
enum Name {value0, value1, ..., valueN};
```

```
enum Suit {CLUBS, DIAMONDS, HEARTS, SPADES};
...
Suit s = CLUBS;
if (s != DIAMONDS) { ... }
```

Ternary operator ? :

```
test ? value1 : value2
```

- **ternary operator**: one that involves 3 operands
  - if test is `true`, evaluates to `value1`; evaluates to `value2`
  - similar to an `if/else` statement, but at the expression level (rather than a block of statements)

- Example:

```
int min(int a, int b) {
    return (a < b) ? a : b;
}
```

Templates

- **template**: A function or class that accepts a type as a parameter
  - allows it to work with many different data types

```
// defining a template function
template <class Name> returntype name(parameters) {
    statements;
}
```

```
// calling a template function
name<TypeName>(parameters);
```

Example:

```
// Returns the lesser of two values.  If equal, returns B.
template <class T> T min(T a, T b) {
    return (a < b) ? a : b;
}

Date sooner = min<Date>(date1, date2);
```

STL

- **Standard Template Library (STL)**: a library of C++ data structures
  - vector, list, stack, queue, priority queue, set, bit set, map, multi-map
  - equivalent to Java Collections API; strives for flexibility and speed

- STL also includes:
  - Iterators: iterate over objects in containers
  - Algorithms: do something to each object in a containers (sort, etc.)
  - Other stuff: function objects, smart "auto" pointers, etc.

- flaws / potential drawbacks
  - STL collections store copies of elements instead of references
  - STL produces long/terrible compiler error messages

STL code example

```
#include <vector>
using namespace std;

vector<Point> v;
v.push_back(Point(3, 5));
v.push_back(Point(4, 7));
...

for (vector<Point>::iterator iter = v.begin();
     iter != v.end();
     ++iter) {
   Point& p = *iter;
   cout << p;
}
```