

CS 485  
Advanced Object Oriented Design

Templates and STL

Spring 2017

<http://www.netobjectives.com/PatternRepository/index.php?title=PatternsByAlphabet>

<http://www.netobjectives.com/files/books/dpe/design-patterns-matrix.pdf>

# Templates

- Generic programming for C++
  - write a chunk of code that can work with any\* type
  - specify the types to use when the code is *invoked*
- Original Purpose:
  - type safe containers (vector, list, ...)
  - Specify the type the container holds and make sure the type is used correctly.
  - get rid of void\* and lots of unsafe casting
- Standard Template Library
  - containers / algorithms / iterators / functors
  - HP->SGI->Many

\* the type must support the operations used in the chunk of code.

# Templates - compile time!

- Inheritance and OO design is usually about dynamic runtime behaviors
- Templates are resolved at compile time

```
class A
{
    public:
        virtual int foo();
        int bar(int x);
        int bar(double x);
};

class B : public A
{
    public:
        virtual int foo();
        int bar(int x);
        int bar(double x);
};

// when are decisions made?
A cAObj;
B cBObj;

A *pcAPtr = new B();
B *pcBPtr = new B();

cAObj.bar(1);
cBObj.bar(1.1);

pcAPtr->foo();
pcBPtr->foo();
```

# Templates

- Function Template

```
template<class T>
T sumThem (const T &lhs, const T &rhs)
{
    return lhs + rhs;
}
```

```
std::cout << sumThem (1, 2) << std::endl;
std::cout << sumThem (1.1, 2.2) << std::endl;
std::cout << sumThem ('a', 1) << std::endl; // type mismatch
std::cout << sumThem ("test", "test") << std::endl; // const char *
std::cout << sumThem (std::string ("CS"), std::string ("485")) << std::endl;
```

# Container Class

## Standard Template Library

```
#include <vector> // backed by dynamic array

std::vector<ExampleClass*> cVectorOfPtrs;

for (int i = 0; i < 10; i++)
{
    cVectorOfPtrs.push_back (new ExampleClass (i, std::to_string (i)));
}

for (int i = 0; i < cVectorOfPtrs.size (); ++i)
{
    std::cout << *cVectorOfPtrs[i] << std::endl;
}
```

# Container

- Type-safe containers
- Sequence Containers
  - maintain ordering - vector
  - iterator
- Associative Containers
  - map or set (ordered or unordered)
  - iterator
- Container Adapters
  - different interface for sequence containers
  - no iterators
  - queue/stack/priority queue

<https://msdn.microsoft.com/en-us/library/1fe2x6kt.aspx>

<http://www.cplusplus.com/reference/stl/>

<http://en.cppreference.com/w/cpp/concept/Container>

# STL

- `<list>`
  - very similar to `<vector>` but backed by a linked list
- `<map>`
  - red/black tree (usually)
  - can visit nodes in order
- `<unordered_map>`
  - hash table
- `<pair>`
- `<iterator>`

# Access Methods

```
// http://en.cppreference.com/w/cpp/language/range-for
```

```
for (auto const &value : cVectorOfPtrs)
```

```
{
```

```
    std::cout << *value << std::endl;
```

```
}
```

```
for (std::vector<ExampleClass*>::iterator it = cVectorOfPtrs.begin ();
```

```
    it != cVectorOfPtrs.end (); ++it)
```

```
{
```

```
    // print pointer value
```

```
    std::cout << *it << std::endl;
```

```
}
```

```
for (auto it = cVectorOfPtrs.begin ();
```

```
    it != cVectorOfPtrs.end (); ++it)
```

```
{
```

```
    std::cout << **it << std::endl;
```

```
    delete *it;
```

```
}
```



# std::for\_each

```
#include <algorithm>

// walk the vector and display each item using for_each
std::for_each (cVectorOfPtrs.begin (),
              cVectorOfPtrs.end (),
              [](auto cExC1)
              {
                std::cout << "FOR_EACH: " << *cExC1 << std::endl;
              });
```

# Objects, not pointers

```
std::vector<ExampleClass> cVector;  
  
for (int i = 10; i < 20; i++)  
{  
    cVector.push_back (ExampleClass (i, std::to_string (i)));  
}
```

# <map>

```
std::map<int, ExampleClass> cTheMap;
for (int i = 20; i < 30; i++)
{
    cTheMap.insert (std::make_pair (i, ExampleClass (i, std::to_string (i))));
}

for (auto it = cTheMap.begin ();
     it != cTheMap.end (); ++it)
{
    std::cout << it->first << " " << it->second << std::endl;
}

//<functional>
std::map<int, ExampleClass, std::greater<int>> cTheMapDescending;
for (int i = 30; i < 40; i++)
{
    cTheMapDescending.insert (std::make_pair (i,
        ExampleClass (i, std::to_string (i))));
}
```

# <map>

```
template<class T>
struct CompareStruct
{
    bool operator()(const T &lhs, const T &rhs) const
    {
        return lhs > rhs;
    }
};
```

```
std::map<std::string, ExampleClass, struct CompareStruct<std::string>>
    cTheMapStringsCompareStruct;
for (int i = 40; i < 50; i++)
{
    cTheMapStringsCompareStruct.insert (std::make_pair
    (std::to_string (rand ()), ExampleClass (i, std::to_string (i))));
}
```

# <unordered\_map>

```
//unordered_map
// http://en.cppreference.com/w/cpp/container/unordered\_map
// hash table
std::unordered_map<int, ExampleClass> cTheHashMap;
for (int i = 50; i < 60; i++)
{
    cTheHashMap.insert (std::make_pair (i, ExampleClass
        (i, std::to_string (i))));
}

auto hashFunc = [](auto &key) { return key + 1; };
std::unordered_map<int, ExampleClass, decltype(hashFunc)>
    cTheHashMapHashGiven (1024, hashFunc);
for (int i = 60; i < 70; i++)
{
    cTheHashMapHashGiven.insert (std::make_pair (i,
        ExampleClass (i, std::to_string (i))));
}
```