CSI33 Data Structures

Department of Mathematics and Computer Science
Bronx Community College

November 22, 2017

Outline

1. Chapter 12: C++ Templates
   - Template Functions
   - Template Classes

Outline

1. Chapter 12: C++ Templates
   - Template Functions
   - Template Classes

Templates Allow Code For Different Types

Python doesn't associate types with variable names, so the same code might work for different types. In this example, the function `Maximum` will find the larger of two numbers having the same type (as long as the operator > is defined for that type). For example, the types `int`, `float`, and even `Rational` will work here:

```
def Maximum(a, b):
   if a > b:
      return a
   else:
      return b
```

C++: Different Versions For Different Types

```
int maximum_int(int a, int b)
{
   if (a > b){
      return a;
   }
   else {
      return b;
   }
}
```

C++: Different Versions For Different Types

```
int maximum_double(double a, double b)
{
   if (a > b){
      return a;
   }
   else {
      return b;
   }
}
```

Template Function Example: C++

```
template <typename Item>

Item maximum(Item a, Item b) {
   if (a > b) {
      return a;
   }
   else {
      return b;
   }
}
```

Template Function Example: C++

```
int main()
{
   int a=3, b=4;
   double x=5.5, y=2.0;
   cout << maximum(a, b) << endl;
   cout << maximum(x, y) << endl;
   return 0;
}
```

C++ Template Classes: Container Classes

- Container classes which provide certain access to each item (stack, Queue, ...) all behave the same for different data types of the items contained.
- Iterators should be provided to allow abstract traversal (without needing to know how the container is implemented).
- C++ template classes are able to provide this.

C++ Template Classes: Container Classes

- As each container class is used for some datatype, the compiled template class for that type is instantiated.
- No code for a template class instance is compiled until it is needed.

The Standard Template Library

- The STL implements most of the common container classes as C++ template classes.
- It is now a standard part of the C++ library.
- It defines a wide variety of containers for classes which implement a few basic operations. (For example, < for binary search trees or priority queues.)
- It provides iterators for these classes.

The vector Template Class

```
int main()
{
   vector<int> iv;
   vector<double> dv;
   int i;
   for (i=0; i<10; ++i) {
      iv.push_back(i);
      dv.push_back(i + 0.5);
   }
   for (i=0; i< 10; ++i) {
      cout << iv[i] << " " << dv[i] << endl;
   }
   return 0;
}
```

The vector Template Class

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
   vector<int> iv(5, 3);
   vector<double> dv(5);
   int i;
   for (i=0; i<5; ++i) {
      cout << iv[i] << " " << dv[i] << endl;
   }
}
```

The vector Template Class

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
   vector<int> iv;
   vector<int>::iterator iter;
   int i;
   for (i=0; i<10; ++i) {
      iv.push_back(i);
   }
   for (iter=iv.begin(); iter != iv.end(); ++iter) {
      cout << *iter << endl;
   }
   return 0;
```

User-Defined Template Classes

- The header file, `<classname>.h`, is the same for ordinary classes, but class definition has a template data type, a "wild card" typename instead of a normal type like `int` or `double`.
- The class definition is preceded by
  `template <typename T>`
  where `T` can be any identifier not in use. (for example, `Item` in the `stack` class.)
- Whenever the template data type is needed in a function declaration, it is used like an ordinary type name:
  `bool pop(Item &item);`
- The last line of the header file includes the implementation file: `#include "<classname>.template"`
  (which does **not** include the header file).

User-Defined Template Classes

```
template <typename Item>
class Stack {
public:
   Stack();
   ~Stack();
   int size() const { return size_; }
   bool top(Item &item) const;
   bool push(const Item &item);
   bool pop(Item &item);
```

User-Defined Template Classes

```
private:
   Stack(const Stack &s);
   void operator=(const Stack &s);
   void resize();
   Item *s_;
   int size_;
   int capacity_;
};
#include "Stack.template"
```

User-Defined Template Classes

```
// Stack.template
template <typename Item>
Stack<Item>::Stack()
{
   s_ = NULL;
   size_ = 0;
   capacity_ = 0;
}
template <typename Item>
Stack<Item>::~Stack()
{
   delete [] s_;
}
```

User-Defined Template Classes

```
// test_Stack.cpp
#include "Stack.h"
int main()
{
   Stack<int> int_stack;
   Stack<double> double_stack;
   int_stack.push(3);
   double_stack.push(4.5);
   return 0;
}
```