ECE 3574: Applied Software Design: Static Polymorphism using Templates

Chris Wyatt

Today we will look at how to reuse code using polymorphism and specifically static polymorphism through generic programming

- Generics in C++ using Templates
- Static Polymorphism
- Exercise 04: How does std::vector work?

Generics in C++

- Templates elevate types to be generic, named but unspecified, and can work with functions and classes.
- Templates allow code reuse as long as the types meet the functionality required by the template
- The C++ standard library uses templates extensively

Example 1: template function to swap

A simple example is a function to swap the contents of two variables (similar to std::swap):

```
template< typename T >
void swap(T& a, T & b)
{
  T temp(b);
  b = a;
  a = temp;
}
```

Example 1: template function to swap

The symbol `T` acts like a variable, in fact it is a type variable. Defined this way swap is generic, I can use it on any type that can be copied. For example:

```
int a = 1;
int b = 2;

std::cout << a << ", " << b << std::endl;
swap(a,b);
std::cout << a << ", " << b << std::endl;

std::string A = "foo";
std::string B = "bar";

std::cout << A << ", " << B << std::endl;
swap(A,B);
std::cout << A << ", " << B << std::endl;
```

Example 1: template function to swap

If the type does not support a particular usage it generates a compile time error. For example suppose I wrote a class that explicitly does not allow copies

```
class NoCopy
{
public:
  NoCopy() = default;
  NoCopy(const NoCopy & x) = delete;
};
```

and tried to use swap as

```
NoCopy x,y;
swap(x,y);
```

My compiler complains

```
swapexample.cpp:7:5: error: call to deleted constructor of
  T temp(b);
```

Example 2: template class to hold a pair of objects

Templates work with classes as well. For example, we might define a tuple holding two different types (aka std::pair) as

```
template <typename T1, typename T2>
class pair
{
public:

  pair(const T1 & first, const T2 & second);

  T1 first();
  T2 second();

private:
  const T1 m_first;
  const T2 m_second;
};
```

Example 2: template class to hold a pair of objects

And implement it like

```
template <typename T1, typename T2>
pair<T1,T2>::pair(const T1 & first, const T2 & second)
: m_first(first), m_second(second)
{}

template <typename T1, typename T2>
T1 pair<T1,T2>::first()
{
  return m_first;
}

template <typename T1, typename T2>
T2 pair<T1,T2>::second()
{
  return m_second;
}
```

Example 2: template class to hold a pair of objects

We might use it like so

```
pair<int,std::string> x(0, std::string("hi"));

std::cout << "First = " << x.first() << std::endl;
std::cout << "Second = " << x.second() << std::endl;
```

Organizing Template Code

The full implementation of a template must occur in the same translation unit. Thus they cannot be compiled and linked separately.

- We still would like to organize our code into a separate definition (header, .h) and implementation file (.cpp)
- Just include the implementation file at the bottom of the header file
- To prevent confusion the implementation file is often given a different extension (.tpp or .txx).

Exercise 04: How does std::vector work?

See website.

Next Actions and Reminders

- Read through a C++ standard library containers reference
- Project 0 is due tomorrow