Object-Oriented Programming (OOP)
Lecture No. 34

Generic Algorithms

A Case Study

Print an Array

```
template< typename T >
void printArray( T* array, int size )
{
  for ( int i = 0; i < size; i++ )
     cout << array[ i ] << ", ";
}
```

Generic Algorithms

```
const int* find( const int* array,
                  int _size, int x ) {
  const int* p = array;
  for (int i = 0; i < _size; i++) {
     if ( *p == x )
          return p;
     p++;
  }
  return 0;
}
```

...Generic Algorithms

```
template< typename T >
T* find( T* array,int _size,
                       const T& x ) {
  T* p = array;
  for (int i = 0; i < _size; i++) {
     if ( *p == x )
          return p;
     p++;
  }
  return 0;
}
```

...Generic Algorithms

```
template< typename T >
T* find( T* array, T* beyond,
                             const T& x ) {
  T* p = array;
  while ( p != beyond ) {
     if ( *p == x )
          return p;
     p++;
  }
  return 0;
}
```

...Generic Algorithms

```
template< typename T >
T* find( T* array, T* beyond,
                            const T& x ) {
  T* p = array;
  while ( p != beyond ) {
     if ( *p == x )
          return p;
     p++;
  }
  return beyond;
}
```

...Generic Algorithms

```
template< typename T >
T* find( T* array, T* beyond,
                       const T& x ) {
  T* p = array;
  while ( p != beyond && *p != x )
     p++;
  return p;
}
```

…Generic Algorithms

- This algorithm works fine for arrays of any type
- We can make it work for any generic container that supports two operations
  - Increment operator (++)
  - Dereference operator (*)

…Generic Algorithms

```
template< typename P, typename T >
P find( P start, P beyond,
     const T& x ) {
  while ( start != beyond &&
                *start != x )
          start++;
  return start;
}
```

…Generic Algorithms

```
int main() {
  int iArray[5];
  iArray[0] = 15;
  iArray[1] = 7;
  iArray[2] = 987;
  …
  int* found;
  found = find(iArray, iArray + 5, 7);
  return 0;
}
```

Class Templates

- A single class template provides functionality to operate on different types of data
- Facilitates reuse of classes
- Definition of a class template follows
  - template< class T > class Xyz { … }; or
  - template< typename T > class Xyz { … };

Example – Class Template

- A **`Vector`** class template can store data elements of different types
- Without templates, we need a separate **`Vector`** class for each data type

…Example – Class Template

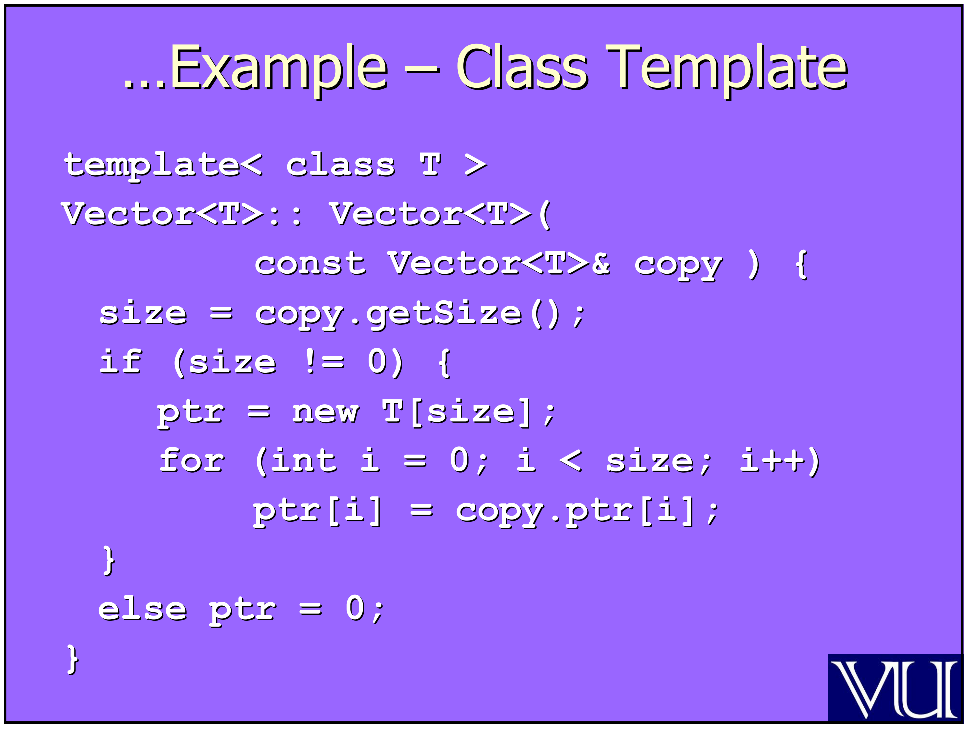
```
template< class T >
class Vector {
private:
  int size;
  T* ptr;
public:
  Vector<T>( int = 10 );
  Vector<T>( const Vector< T >& );
  ~Vector<T>();
  int getSize() const;
```

…Example – Class Template

```
  const Vector< T >& operator =(
                  const Vector< T >& );
  T& operator []( int );
};
```

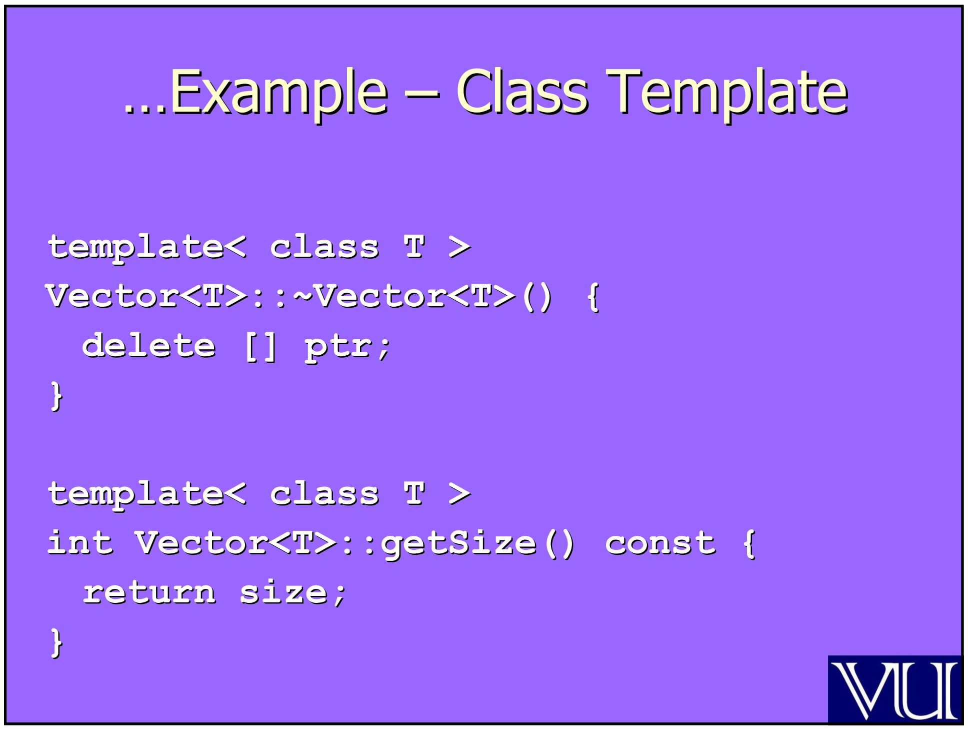
…Example – Class Template

```
template< class T >
Vector<T>::Vector<T>( int s ) {
  size = s;
  if ( size != 0 )
     ptr = new T[size];
  else
     ptr = 0;
}
```



…Example – Class Template

```
template< class T >
Vector<T>:: Vector<T>(
          const Vector<T>& copy ) {
  size = copy.getSize();
  if (size != 0) {
     ptr = new T[size];
     for (int i = 0; i < size; i++)
          ptr[i] = copy.ptr[i];
  }
  else ptr = 0;
}
```



…Example – Class Template

```
template< class T >
Vector<T>::~Vector<T>() {
  delete [] ptr;
}

template< class T >
int Vector<T>::getSize() const {
  return size;
}
```

…Example – Class Template

```
template< class T >
const Vector<T>& Vector<T>::operator
          =( const Vector<T>& right) {
  if ( this != &right ) {
     delete [] ptr;
     size = right.size;
```

…Example – Class Template

```
     if ( size != 0 ) {
          ptr = new T[size];
          for(int i = 0; i < size;i++)
               ptr[i] = right.ptr[i];
     }
     else
          ptr = 0;
  }
  return *this;
}
```

…Example – Class Template

```
template< class T >
T& Vector< T >::operator [](
                              int index ) {
  if ( index < 0 || index >= size ) {
      cout << "Error: index out of
                              range\n";
      exit( 1 );
  }
  return ptr[index];
}
```

…Example – Class Template

- A customization of above class template can be instantiated as

```
Vector< int > intVector;
…
Vector< char > charVector;
```