

# EECS402 Lecture 17

Andrew M. Morgan

Savitch Ch. 16 Templates

```
Consider This Program
     int main (void)
       int i1 = 5, i2 = 8, i3, i4;
float f1 = 19.2, f2 = -2.3, f3, f4;
       char c1 = 'h', c2 = 'p', c3, c4;
       BlipClass b1(6, 9), b2(7, 7), b3, b4;
                                                     Ints: 58
       i3 = lesser(i1, i2);
                                                     Floats: -2.3 19.2
       i4 = greater(i1, i2);
                                                     Chars: h p
       f3 = lesser(f1, f2);
                                                     Blips: blip: 77 blip: 69
       f4 = greater(f1, f2);
       c3 = lesser(c1, c2);
       c4 = greater(c1, c2);
       b3 = lesser(b1, b2);
       b4 = greater(b1, b2);
       cout << "Ints: " << i3 << " " << i4 << endl;
       cout << "Floats: " << f3 << " " << f4 << endl;
       cout << "Chars: " << c3 << " " << c4 << endl;
       cout << "Blips: " << b3 << " " << b4 << endl;
       return 0;
EECS
                                Andrew M Morgan
402
```

```
EEC:
                      Functions We Need
float lesser(
                  int lesser(
                                   char lesser(
                                                    BlipClass lesser(
                                       char v1,
     float v1,
                     int v1,
                                                         BlipClass v1,
     float v2)
                     int v2)
                                       char v2)
                                                         BlipClass v2)
 float res;
                   int res;
                                    char res;
                                                      BlipClass res;
 if (v1 < v2)
                   if (v1 < v2)
                                    if (v1 < v2)
                                                      if (v1 < v2)
                                                       res = v1;
   res = v1;
                    res = v1;
                                     res = v1;
 else
                   else
                                    else
                                                      else
   res = v2;
                     res = v2;
                                      res = v2;
                                                        res = v2;
 return res;
                   return res;
                                    return res;
                                                      return res;
float greater (
                 int greater(
                                                    BlipClass greater (
                                  char greater (
     float v1,
                     int v1,
                                       char v1,
                                                         BlipClass v1,
                                                         BlipClass v2)
     float v2)
                     int v2)
                                       char v2)
 float res;
                   int res;
                                                     BlipClass res;
                                    char res;
                   if (v1 > v2)
 if (v1 > v2)
                                   if (v1 > v2)
                                                     if (v1 > v2)
   res = v1;
                   res = v1;
                                     res = v1;
                                                      res = v1;
                   else
                                    else
                                                      else
   res = v2;
                   res = v2;
                                     res = v2;
                                                      res = v2;
 return res;
                   return res;
                                    return res;
                                                      return res;
EECS
                             Andrew M Morgan
402
```

# What A Pain – Eh? Notice that all the functions are essentially the same The only thing that changes is the data type that the function performs on Wouldn't it be nice if we could use a variable to describe the type? Obviously this can not be done, because we would have some sort of assignment like "myType = float;" – This is obviously NOT valid C++ syntax However, we can achieve similar functionality using templates in C++



### **Introduction To Templates**

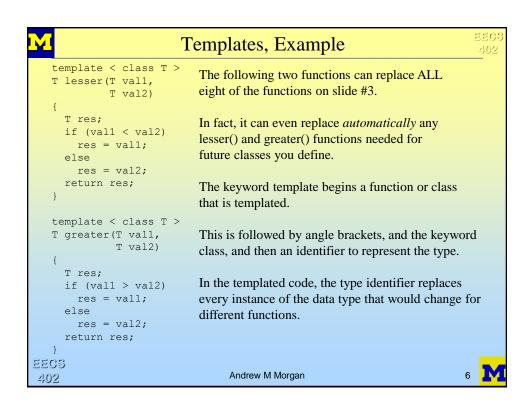


- A template is a special construct in C++ that allows multiple types to be operated on by a single implementation of a function
- Think of a template as describing an algorithm
  - Since the operation we are performing is performed the same way for all different data types, it is sufficient to give code to implement an algorithm without being specific to a data type
- The word template is a keyword in C++
- A function can be "templated" as would be the case for the lesser() and greater() functions
- A class can also be "templated"
- A templated function or class can have different data types that are left unspecified
  - In other words, multiple types can be templated

**全国の** 402

Andrew M Morgan







### **Using Templates**

- How do you use a templated function though?
- Just as you would any other function
- The hard work is done by the compiler
  - When passing through your program, the compiler detects which types are needed by a templated function.
  - The compiler then automatically generates a separate function for each type needed
  - The function is then compiled and is available for use, without any required actions by the user
- A programmer using a templated function may not even realize it was templated

EECS 703

Andrew M Morgan



```
Using Templates, Example
     int main (void)
                                                   This is the exact same
                                                   main() function shown
       int i1 = 5, i2 = 8, i3, i4;
                                                   earlier, when all 8
       float f1 = 19.2, f2 = -2.3, f3, f4;
                                                   individual lesser() and
       char c1 = 'h', c2 = 'p', c3, c4;
                                                   greater() functions were
       BlipClass b1(6, 9), b2(7, 7), b3, b4;
                                                   implemented.
       i3 = lesser(i1, i2);
                                                   Currently, the BlipClass
       i4 = greater(i1, i2);
       f3 = lesser(f1, f2);
                                                   operations are
       f4 = greater(f1, f2);
                                                   commented. This is
       c3 = lesser(c1, c2);
                                                   explained in the next 4
       c4 = greater(c1, c2);
                                                   slides.
       //b3 = lesser(b1, b2);
       //b4 = greater(b1, b2);
       cout << "Ints: " << i3 << " " << i4 << endl;
                                                             Ints: 58
       cout << "Floats: " << f3 << " " << f4 << endl;
                                                             Floats: -2.3 19.2
       cout << "Chars: " << c3 << " " << c4 << endl;
                                                             Chars: h p
       //cout << "Blips: " << b3 << " " << b4 << endl;
       return 0;
EECS
                                Andrew M Morgan
402
```



### **Assumption Made**



- Very important note: What assumptions are made when you make a templated function?
- While you originally write lesser() and greater() for integers, or floats, etc, it may be used for other types
- For example, the two functions were used on the type "BlipClass" which was a user-defined class
- You MUST be sure that your user-defined classes provide any operations and/or operators that are used in the templated function
- That means the class blip must have defined an implementation for determining what ">" and "<" means for these blip objects</li>

EECS 402

Andrew M Morgan





## More On Template Assumptions

402 402

Consider this implementation of the BlipClass:

```
class BlipClass
{
  public:
    int v1;
    int v2;

    BlipClass():v1(0),v2(0) { ; }
    BlipClass(int in1, int in2):v1(in1),v2(in2) { ; }
};
```

The following results when trying to compile the previous program:

```
Insertion operator
BlipClass.cpp: In function 'int main()':
                                                                                            used in main()
BlipClass.cpp:57: no match for `ostream & << BlipClass &'
BlipClass.cpp: In function `class BlipClass lesser<BlipClass>(BlipClass, BlipClass)':
                                                                                         Less-than operator
BlipClass.cpp:51: instantiated from here
                                                                                          used in templated
                                                                                              lesser()
BlipClass.cpp:9: no match for `BlipClass & < BlipClass &'
BlipClass.cpp: In function 'class BlipClass greater<BlipClass>(BlipClass, BlipClass)':
                                                                                         Greater-than operator
BlipClass.cpp:52: instantiated from here
                                                                                          used in templated
BlipClass.cpp:21: no match for `BlipClass & > BlipClass &'
                                                                                              greater()
EECS
                                             Andrew M Morgan
402
```

```
Updated BlipClass Implementation
class BlipClass
                                                 Added functionality allows earlier
                                                 main() to compile and be executed.
  public:
    int v1;
    int v2;
    BlipClass():v1(0),v2(0) { ; }
    BlipClass(int in1, int in2):v1(in1),v2(in2) { ; }
    bool operator<(const BlipClass &rhs)
      return (v1 + v2) < (rhs.v1 + rhs.v2);
    bool operator>(const BlipClass &rhs)
      return (v1 + v2) > (rhs.v1 + rhs.v2);
};
ostream &operator << (ostream &os, const BlipClass &rhs)
  os << "blip: " << rhs.v1 << " " << rhs.v2;
  return os;
EECS
                                Andrew M Morgan
703
```

```
Final Execution of main() Function
     int main (void)
       int i1 = 5, i2 = 8, i3, i4;
       float f1 = 19.2, f2 = -2.3, f3, f4;
       char c1 = 'h', c2 = 'p', c3, c4;
                                                   Ints: 58
       BlipClass b1(6, 9), b2(7, 7), b3, b4;
                                                   Floats: -2.3 19.2
                                                   Chars: h p
       i3 = lesser(i1, i2);
                                                   Blips: blip: 7 7 blip: 6 9
       i4 = greater(i1, i2);
       f3 = lesser(f1, f2);
       f4 = greater(f1, f2);
                                         Note: Absolutely no change from
       c3 = lesser(c1, c2);
       c4 = greater(c1, c2);
                                         the example using non-templated
       b3 = lesser(b1, b2);
                                         functions...
       b4 = greater(b1, b2);
       cout << "Ints: " << i3 << " " << i4 << endl;
       cout << "Floats: " << f3 << " " << f4 << endl;
       cout << "Chars: " << c3 << " " << c4 << endl;
       cout << "Blips: " << b3 << " " << b4 << endl;
       return 0;
EECS
                               Andrew M Morgan
402
```



### **Templated Classes**

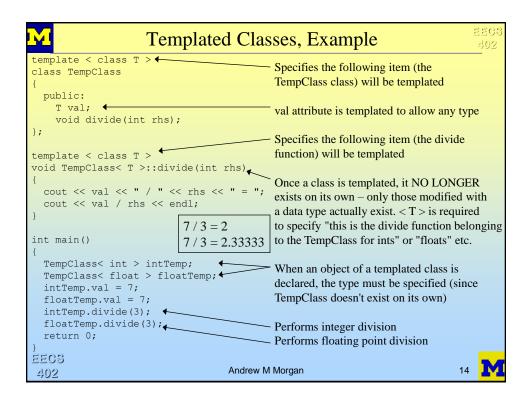
트트(5) 402

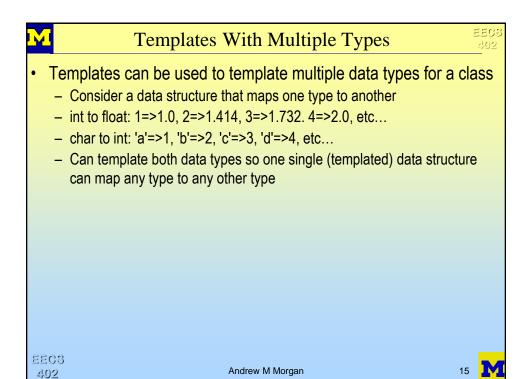
- Templated classes are a bit different than templated functions
- The main reason for this is because a class is a type and objects will be declared of that type
  - Therefore, you must know what type(s) of data will be stored in the object you are creating
- This means, while a programmer may not know a function is templated, they must know a class is templated
- When creating an object of a templated class, the programmer must provide the data type during the declaration

로르CS 402

Andrew M Morgan

N





```
Templating Multiple Data Types, Example
      template <class DT1, class DT2>
      class DataWithKey
          DT1 theKey; //This is the key associated with the item.
                   //This is the data value of the item.
          DT2 val;
        public:
          DataWithKey() { ; }
          DataWithKey(DT1 inKey, DT2 inVal):theKey(intKey), val(inVal) { ; }
          //Reader / writer functions here..
          DT1 getKey()
          { return theKey; }
          DT2 getVal()
          { return val; }
          void setInfo(DT1 inKey, DT2 inVal)
            theKey = inKey;
            val = inVal;
          //ASSUMPTION: The << must be available for both the type of the
                      value and the type of the key!
          void Print()
            cout << " Data: " << val << endl;
           cout << " Key: " << theKey << endl;</pre>
EECS
                                Andrew M Morgan
402 };
```

```
Using The Multiply Templated Class
       int main()
                                                          Student #: 4819 Name: Homer
         int i:
                                                          Student #: 9811 Name: Marge
         DataWithKey< int, string > students[4];
                                                          Student #: 1624 Name: Bart
         DataWithKey< char, float > gradePerc[4];
                                                          Student #: 3902 Name: Lisa
                                                          Grade: A Min %age: 88.5
         students[0].setInfo(4819, "Homer");
         students[1].setInfo(9811, "Marge");
students[2].setInfo(1624, "Bart");
                                                          Grade: B Min %age: 75
                                                          Grade: C Min %age: 64.5
         students[3].setInfo(3902, "Lisa");
                                                          Grade: D Min %age: 56.25
         for (i = 0; i < 4; i++)
           cout << "Student #: " << students[i].getKey() <<</pre>
                 " Name: " << students[i].getVal() << endl;
         gradePerc[0].setInfo('A', 88.5);
         gradePerc[1].setInfo('B', 75.0);
         gradePerc[2].setInfo('C', 64.5);
         gradePerc[3].setInfo('D', 56.25);
         for (i = 0; i < 4; i++)
           cout << "Grade: " << gradePerc[i].getKey() <<</pre>
                   " Min %age: " << gradePerc[i].getVal() << endl;
         return 0;
EECS }
                                     Andrew M Morgan
703
```

