CS 106X, Lecture 17
Advanced Classes

reading:

Programming Abstractions in C++, Chapters 6, 11.3, 14

Plan For Today

- Templates
- Operator Overloading
- Announcements

Learning Goals

- Understand how to make generic classes using templates
- Understand how to overload operators to use with your custom variable types
- Understand how to use C++ arrays

Plan For Today

- Templates
- Operator Overloading
- Announcements

Template function (14.1-2)

```
template<typename T>
returntype name(parameters) {
    statements;
}
```

- **Template**: A function or class that accepts a *type parameter(s)*.
  - Allows you to avoid redundancy by writing a function that can accept many types of data.
  - Templates can appear on a single function, or on an entire class

Template func example

```
template<typename T>
T max(T a, T b) {
    if (a < b) { return b; }
    else       { return a; }
}
```

– The template is *instantiated* each time you use it with a new type.
  - The compiler actually generates a new version of the code each time.
  - The type you use must have an operator < to work in the above code.

```
int i    = max(17, 4);           // T = int
double d = max(3.1, 4.6);        // T = double
string s = max(string("hi"),     // T = string
               string("bye"));
```

Template class (14.1-2)

- **Template class**: A class that accepts a type parameter(s).
  - In the header and cpp files, mark each class/function as templated.
  - Replace occurrences of the previous type `int` with T in the code.

```
// ClassName.h
template<typename T>
class ClassName {
    ...
};

// ClassName.cpp
template<typename T>
type ClassName::name(parameters) {
    ...
}
```

Template .h and .cpp

- Because of an odd quirk with C++ templates, the separation between .h header and .cpp implementation must be reduced.
  - Either write all the bodies in the .h file (suggested),
  - Or `#include` the .cpp at the end of .h file to join them together.

```
// ClassName.h
#ifndef _classname_h
#define _classname_h

template<typename T>
class ClassName {
    ...
};

#include "ClassName.cpp"
#endif   // _classname_h
```

Exercise

- Convert the **`LinkedListClass`** to use templates.
  - A client should be able to create a LinkedListClass of any type.

```
LinkedListClass<int> s1;
s1.add(42);
s1.add(17);

LinkedListClass<string> s2;
s2.add("hello");
s2.add("there");
...
```

Plan For Today

- Templates
- Operator Overloading
- Announcements

Operator overloading (6.2)

- **operator overloading**: Redefining the behavior of a common operator in the C++ language.

  unary: `+ - ++ -- * & ! ~ new delete`

  binary: `+ - * / % += -= *= /= %= & | && || ^ == != < > <= >= = [] -> () ,`

- Syntax:

```
returnType operator op(parameters);        // .h

returnType operator op(parameters) {       // .cpp
    statements;
};
```

  – the ***parameters*** are the operands next to the operator;
  for example, `a + b` becomes `operator +(Foo a, Foo b)`

Op overload example

```
// BankAccount.h
class BankAccount {
   ...
};

bool operator ==(const BankAccount& ba1,
                 const BankAccount& ba2);
```

```
// BankAccount.cpp
bool operator ==(const BankAccount& ba1,
                 const BankAccount& ba2) {
    return ba1.getName() == ba2.getName()
        && ba1.getBalance() == ba2.getBalance();
}
```

Make objects printable

- To make it easy to print your object to `cout`, overload the `<<` operator between an `ostream` and your type:

```
ostream& operator <<(ostream& out, Type& name) {
    statements;
    return out;
}
```

  - `ostream` is a class that represents `cout`, file output streams, etc.

  - The operator returns a reference to the stream so it can be chained.
    - `cout << a << b << c` is really `((cout << a) << b) << c`
    - Technically `cout` is being returned by each `<<` operation.

<< overload example

```
// BankAccount.h
class BankAccount {
    ...
};

ostream& operator <<(ostream& out, BankAccount& ba);
```

```
// BankAccount.cpp
ostream& operator <<(ostream& out, BankAccount& ba) {
    out << ba.getName() << ": $"
        << fixed << setprecision(2)
        << ba.getBalance();
    return out;
}
```

Alternate syntax

- You can also declare operators inside the class.
  - The `this` object is implicitly the first parameter.
  - The internal operator can access the objects' private data.

```
// BankAccount.h
class BankAccount {
    bool operator ==(const BankAccount& ba2);
};
```

```
// BankAccount.cpp
bool BankAccount::operator ==(const BankAccount& ba2) {
    return name == ba2.name && balance == ba2.balance;
}
```

Plan For Today

- Templates
- Operator Overloading
- Announcements

Announcements

- CS198 Section Leading Application due Fri. Nov 2
  - See cs198.stanford.edu
- Ethics, Technology, and Public Policy workshops Fri. Nov 2
  - Run by Mehran Sahami, Rob Reich, Jeremy Weinstein
  - Two interactive workshops
  - Register here: https://docs.google.com/forms/d/e/1FAIpQLSfeaMjuse6LE-3vk3cupkZ6uVJufluQKu5QD-5jazgpbOgEYg/viewform?usp=sf_link

Recap

- Templates
- Operator Overloading
- Announcements

- **Next time:** Arrays, Trees