Lecture 7 Notes

Operator overloading

In C++, specific behavior can be defined for when a particular operator is used with a data type derived from a class. This is accomplished by creating member functions whose names begin with “operator”.

In the following example, the class GeometricSequence overloads the subscript operator.

```
#include <iostream>
#include <cmath>
using namespace std;

class GeometricSequence {
public:
   double a, r;

   double operator[](int i) const
   {
      return a * pow(r, i);
   }
};

int main()
{
   int i;
   GeometricSequence gs = { 2, 3 };

   for (i = 0; i < 10; ++i) {
      cout << gs[i] << endl;
   }
}
```

The class GeometricSequence represents an infinite sequence of numbers where the ith term is equal to ar^i. The value a is the value of the first term, and r is the common ratio between each term.

The program in this example uses the values 2 and 3 for a and r respectively, then it displays the first 10 values from this sequence.

By overloading the subscript operator, the GeometricSequence data type gives the illusion that it stores an infinite sequence of values when in reality only 2 variables are stored by this class.

You can overload almost any operator for your class. Here is another class called IntegerSet which overloads the subscript and function call operators:

```
#include <iostream>
#include <vector>
using namespace std;

```

```
class IntegerSet {
public:
   int operator[](int i) const
   {
      return i;
   }
   vector<int> operator()(int lower, int upper, int step = 1) const
   {
      int i;
      vector<int> v;
      for (i = lower; i <= upper; i += step) {
         v.push_back(i);
      }
      return v;
   }
}

int main()
{
   IntegerSet is;
   vector<int> v;

   v = is(0, 20, 2);
   for (int i : v) {
      cout << i << endl;
   }
}
```

This example displays the even integers from 0 to 20. The IntegerSet class represents the set of all integers. The subscript operator lets you access a single integer from this set and the function call operator extracts a range of integers from this set into a vector.

Constructors

This next class represents a a mathmatical function for a straight line ($y = mx + b$). (See next page)

```
#include <iostream>
using namespace std;

class LinearFunction {
private:
   double slope, bias;
public:
   LinearFunction(double _slope, double _bias)
   {
      slope = _slope;
      bias = _bias;
   }
   double operator()(double x) const
   {
      return slope * x + bias;
   }
};

int main()
{
   LinearFunction f = { 2.0, 3.0 };
   cout << f(5.0) << endl;
}
```

Notice the function named LinearFunction. Any member function that has the same name as it's class is known as a constructor.

The constructor is the first function that executes when a variable of that class gets declared.

Since the only LinearFunction constructor requires 2 parameters, it would be illegal to declare a LinearFunction variable without initializing it; the statement `LinearFunction f;` would result in a compiler error. This is why the variable f is immediately initialized with values 2.0 and 3.0 in the main function.

To allow a LinearFunction to be created without explicit initialization values, we can create a default constructor: (See next page)

```
 1 class LinearFunction {
 2 private:
 3    double slope, bias;
 4 public:
 5    LinearFunction()
 6    {
 7       slope = 1.0;
 8       bias = 0.0;
 9    }
10    LinearFunction(double _slope, double _bias)
11    {
12       slope = _slope;
13       bias = _bias;
14    }
15    double operator()(double x) const
16    {
17       return slope * x + bias;
18    }
19 };
20
21 int main()
22 {
23    LinearFunction f;
24    f = { 2.0, 3.0 };
25    cout << f(5.0) << endl;
26 }
```

The function LinearFunction(), which takes no parameters, is our default constructor. When the variable f is declared (line 23) without initialization values, the default constructor gets executed.

The statement `f = { 2.0, 3.0 };` has the same effect as executing the f's non-default constructor.

– *Mark Swoope*