Welcome

Object Oriented Programming with C++
CIS 265

Week 7 – Operator overloading, STL

Christopher K. Burns

Schedule

Week		Content
1	1/11	*Review* Chapter 1 Intro to Computers and C++ Programming Chapter 2 Control Structures Chapter 3 Functions Chapter 4 Arrays
2	1/18	Additional Array and Function Topics Chapter 5 Pointers and Strings Lab 1 – Functions, Arrays, and Strings Homework 1 Assigned
3	1/25	Chapter 6 Classes and Data Abstraction
4	2/1	Chapter 7 Classes: Part I Lab 2 – Classes 1 Homework 1 Due
5	2/8	***MID-TERM EXAMINATION*** *(Chapters 1 through 7*)* **only portions of chapter 7 that were covered in class* *Final Project Assigned*
6	2/15	Chapter 7 Classes: Part II Chapter 8 Operator Overloading Homework 2 Assigned
7	2/22	Chapter 8 Operator Overloading Homework 2 Due Lab 3 – Classes 2
8	3/1	Chapter 9 Inheritance: Part I Homework 3 Assigned
9	3/8	Chapter 9 Inheritance: Part II Chapter 10 Polymorphism Homework 3 Due
10	3/15	Chapter 10 Polymorphism Lab 4 – Inheritance and Polymorphism
11	3/22	***FINAL PROJECTS DUE***

Agenda

Tonight's agenda

- Final Project Topics
  - Sample Topics
  - Disk System Class
- More Overloading
- STL
  - String
  - Vector
  - Algorithm

Object Oriented Programming

Home Work

Homework 2 – due tonight

Read Chapter 9 in text

Lab 2 & Homework 2 - Review

Final Project

Project Criteria

- You must use at least one C++ class
- Your class must use at least one of the following techniques
  - operator overloading
  - inheritance
  - vector container
- Project is due on March 22

Start soon! There are only 3 weeks left.

Final Project Topics

Choosing a topic –

You can use the project as a chance to create something that is useful to you

If you are having difficulty selecting a topic:

- Go through the chapters that we have, and will cover in the text book
- Find a program that either meets the criteria, or you can extend it to do so

Final Project Topics

Sample topics –

You are encouraged to select your own topic, but here are a few sample topics:

– graphics
  - Although everything we have done uses text outputs, you can create classes to display shapes using text (see fig 9.3)

– data hierarchy
  - Implement a data hierarchy and program to exercise it (see fig 9.2)

– game
  - Card game
  - Dice game

Final Project Topics

Starter Classes –

Storage System class

– Can be used to create a project that manipulates files

OpenGL class

– If you are interested in 3D graphics, this class can help get you started in OpenGL

Black Jack game and card classes

– Set of card classes that can be used to create a text based card game

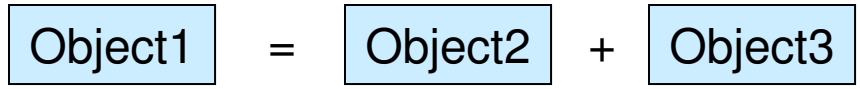
Classes - Review

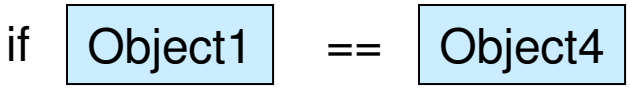
What we've covered about classes up to this point:

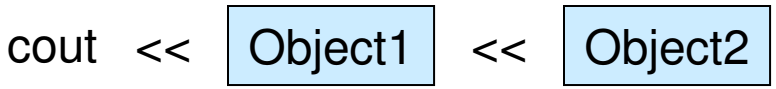
- Basic structure of a class
- Constructors and overloading
- Separating Interface from Implementation
- Data encapsulation (get/set)
- Static members
- this pointer
- Operator Overloading

Operator Overloading

You can overload operators on any class you create, thus allowing objects of your class to be used:







Operator Overloading

Operators that can be overloaded

```
+       -       *       /       %       ^       &       |
~       !       =       <       >       +=      -=      *=
/=      %=      ^=      &=      |=      <<      >>      >>=
<<=     ==      !=      <=      >=      &&      ||      ++
--      ->*     ,       ->      []      ()      new     delete
new[]           delete[]
```

Operators that can not be overloaded

```
.       .*      ::      ?:
```

Operator Overloading

Overloading the [ ] operator:

```
ClassA[ index ]
```

The [ ] operator will take an integer parameter

The [ ] operator must return a reference to ClassA

```
class SomeClass
{
public:
  someType operator::[]( index )
  {
    return someType of value at index
  }
};
```

Operator Overloading

Example: overloading the [ ] operator

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

class Integer
{
private:
  int value;
  char valueString[ 256 ];
  int digits;
public:
  int getDigits() {
    return digits;
  }
  Integer& operator=( int value ) {
    this->value = value;
    itoa( value, this->valueString, 10 );
    this->digits = strlen( this->valueString );
    return *this;
  }
  int operator[]( int index ) {
    return valueString[ index ] - 48;
  }
};
```

```
void main()
{
  Integer integer;
  integer = 459874;
  for ( int i = 0; i < integer.getDigits(); i++ )
  {
    cout << "digit " << i << " = " << integer[i] << endl;
  }
}
```

Operator Overloading

Overloading the ++ operator

There are two forms:

`object++` (postfix)

and

`++object` (prefix)

Operator Overloading

Overloading the ++ operator

`++object` is overloaded as

```
ClassA operator++()
```

`object++` is overloaded as

```
ClassA& operator++( int )
```

Compiler needs to be able to distinguish between these two methods. To do so, the postfix of the increment passes an `int` as a parameter

Operator Overloading

Example: overloading the ++ operators

```
class Integer
{
private:
  int value;
public:
  Integer& operator=( int value )
  {
    this->value = value;
    return *this;
  }
  Integer& operator++() // prefix
  {
    this->value++;
    return *this;
  }
  Integer operator++( int ) // postfix
  {
    Integer temp = *this;
    this->value++;
    return temp;
  }
  friend ostream& operator<< ( ostream& out, Integer printMe )
  {
    out << "Integer is " << printMe.value;
    return out;
  }
};
```

```
void main()
{
  Integer integer;
  integer = 12;
  cout << integer++ << endl;
  cout << ++integer << endl;
}
```

The Standard Library

The C++ Standard Library,

formerly known as STL (Standard Template Library)

Provides many useful structures and containers

Two of these that we are going to look at are the:

`string (basic_string)`

`vector`

A good reference to the STL containers:

http://www.cppreference.com

Standard String

The C++ standard string is a class. It includes numerous methods for

- inserting strings,
- finding substrings,
- testing for empty,
- Overloaded operators such as +=, =, [] to append, assign, and manipulate strings

Standard String

First, a little background

- string class is defined in <string> and is derived from basic_string
- Let's look at the basic string interface defined in <xstring>

Standard String Methods

Basic methods of string

- append (also operator +=)
- assign (also operator =)
- at (also operator [])
- c_str – returns char*
- copy – makes a char array copy of the string
- empty – `if ( myString.empty() )`
- erase – `erases parts of the string`
- length – `cout << myString.length()`

Standard String Methods

Substring manipulation

- insert
- replace
- substr

Standard String Methods

Search methods of string

- compare (also operator ==, !=, >…)
- find
- find_first_of
- find_first_not_of
- find_last_of
- find_last_not_of

Standard String Methods

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

// basic string parsing demo
void main()
{
  string inputString1;
  string inputString2;
  string combinedString;

  // user enters a string
  cout << "Enter word 1:"; cin >> inputString1;
  cout << "Enter word 2:"; cin >> inputString2;

  // overloaded string == operator
  if ( inputString1 == "hello" )
  {
    cout << "aloha" << endl;
  }
  // overloaded string = and + operators
  combinedString = inputString1 + " " + inputString2;

  // print the combined string
  cout << combinedString << endl;
}
```

Standard String Methods

```
#include <iostream>
#include <string>
using namespace std;
// basic string parsing demo
void main()
{
  string inputString;

  // user enters a string
  cout << "Enter a string:";
  cin >> inputString;

  // split string at comma character
  long splitAt;

  // find the position of the first comma
  while( ( splitAt = inputString.find_first_of( ',' ) ) != string::npos )
  {
    // split off the first word seperated by a comma
    string temp = inputString.substr( 0, splitAt );
    // remove this first word from the input string
    inputString = inputString.substr( splitAt + 1, inputString.length() - splitAt - 1 );
    // print out the split off word
    cout << temp << endl;
  }
  // print what is left of the string
  cout << inputString;
}
```

Vector

A vector is a C++ template class that contains a list of values (array) and useful utility methods.

declare a vector:

```
vector< type-of-data > vectorName;
```

example:

```
vector< int >    arrayOfInts;
vector< string > arrayOfStrings;
```

Vector

Vector contains many features that make it much easier, and safer, to use than a standard C type array.

Pros

- Do not need to know the size of the array ahead of time
- Knows how large it is
- Useful methods, such as one swap values

Cons

- Data in vector is not stored contiguously in memory (means you can't use pointer math tricks)

Vectors

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

void main()
{
  vector<int> arrayOfInts;
  string inputString;
  cout << "Enter numbers (enter \"end\" to stop)" << endl;
  while ( true )
  {
    cin >> inputString;
    if ( inputString == "end" )
      break;
    arrayOfInts.push_back( atoi( inputString.c_str() ) );
  }
  for ( int i = 0; i < arrayOfInts.size(); i++ )
  {
    cout << arrayOfInts[ i ] << " ";
  }
  cout << endl;
}
```

Vectors and Algorithms

The Standard library also contains basic algorithms that can be used on vectors

- sort
- find
- random_shuffle
- replace
- reverse
- min_element
- max_element
- copy

Vectors and Algorithms

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


void getList( vector<int>& ints )
{
  string inputString;
  cout << "Enter numbers "
       << "(enter \"end\""
       << "to stop)" << endl;
  while ( true )
  {
    cin >> inputString;
    if ( inputString == "end" )
      break;
    ints.push_back(atoi( inputString.c_str() ));
  }
}

void printList( vector<int> ints )
{
  for ( int i = 0; i < ints.size(); i++ )
  {
    cout << ints[ i ] << " ";
  }
  cout << endl;
}
```

```
// basic string parsing demo
void main()
{
  vector<int> arrayOfInts;
  getList( arrayOfInts );

  cout << "sorted" << endl;
  sort( arrayOfInts.begin(), arrayOfInts.end() );
  printList( arrayOfInts );

  cout << "shuffled" << endl;
  random_shuffle( arrayOfInts.begin(), arrayOfInts.end() );
  printList( arrayOfInts );
}
```

More STL Containers

Besides vectors, a few of the others are:

- **lists** – linked list of elements
- **queues** – FIFO list
- **stacks** – FILO list
- **maps** – quick data lookups (key/value pairs)