More about Classes

COS110: Chapter 14
Dr. Mardé Helbig
Static member variables with constant values

• How can we define a static member variable that has a fixed value?
Memberwise Assignment

• We can assign one object’s data to another object of the same class
• We can also initialize one object with another object’s data
• We use the “=” operator to do this
• By default, each member of one object is copied to the corresponding member in the other object

Example Code
Memberwise Assignment

• What is the problem here? All copying happening is only shallow copies
• Dynamic memory allocation => only the address of pointer will be copied and not the data
• Both object’s variable will now point to the same address
• If one object is destroyed, the destructor is called and that releases the memory! What now?
Memberwise Assignment

What happens if destructor of wizard1 is called?

Can solve it with copy constructor
Copy Constructor

- Constructor that is called if new object is created and initialized with another object’s data
- Used when we have dynamically allocated memory
- Ensures that deep copy is taking place for array member variable
- Take a reference to an object of the same class
- Reference enable data corruption of the object passed as reference
- Can use `const` to prevent this
Copy Constructor

- Object passed by value as argument to function
- Passed to parameter that is also a class object
- Copy constructor of class parameter is called
- Object used as function argument, is created when function is called and it’s initialized with argument’s value
- Therefore C++ requires parameter of copy constructor to be a reference object
Default Copy Constructor

• If copy constructor is not defined for class, C++ creates a default copy constructor
• Default copy constructor performs memberwise assignment => shallow copying
Operator Overloading

• C++ provides many operators to manipulate data of primitive data types
• How can we define these operators for class objects?
• For example:

  Can use: int x=4; int y=5; int z = x+y;

  But, want to use: Wizard wizard1; Wizard wizard2;
  Wizard wizard3 = wizard1 + wizard2;

  Want wizard3 to have all spells of wizard1 and wizard2

• C++ allows you to redefine how standard operators work when used with class objects
• Called operator overloading
Overloading = Operator

- Copy constructor ensures deep copying when a new object is created and initialized with another.
- But it does not deal with normal assignment statements.
- Assignment statements invoke memberwise assignment.

Example: `wizard4 = wizard2;`

- For these cases we must overload the `=` operator (assignment operator).
- Re-define the operator, which is then called whenever the operator is used with an object of class.
Overloading = Operator

• Copy constructor ensures deep copying when a new object is created and initialized with another
• But it does not deal with normal assignment statements
• Assignment statements invoke memberwise assignment

Example: `wizard4 = wizard2;`

• For these cases we must overload the = operator (assignment operator)
• Re-define the operator, which is then called whenever the operator is used with an object of class
Overloading = Operator

void operator = (const Wizard &wizard)

wizard2 = wizard1;

• Parameter is a reference object
• Prevents compiler from making copy of object being passed into function
• const ensures that function does not accidentally change contents of the passed object
Overloading = Operator

• Calling
  `wizard2 = wizard1;`

• Is equal to calling:
  `wizard2.operator = (wizard1);`
Overloading = Operator

class Wizard
{
    void operator = (const Wizard &wizard);
};

void Wizard::operator = (const Wizard &wizard)
{
    delete [] spells;
    name = wizard.name;
    spellsIndex = wizard.spellsIndex;
    spells = new string[wizard.spellsIndex];
    for (int i=0; i < spellsIndex; i++)
        spells[i] = wizard.spells[i];
}

Destroy existing spells array so that a new one can be created that is similar to the object being copied

Rest is similar to copy constructor
class Wizard
{
    void operator = (const Wizard &wizard);
};

void Wizard::operator = (const Wizard &wizard)
{
    delete [] spells;
    name = wizard.name;
    spellsIndex = wizard.spellsIndex;
    spells = new string[spellsIndex];
    for (int i=0; i < spellsIndex; i++)
        spells[i] = wizard.spells[i];
}

Problem: it is not returning anything. So cannot use it for:
wizard1 = wizard2 = wizard3;

Need to return an object to be able to do this
class Wizard
{
    const Wizard operator = (const Wizard &wizard);
};

const Wizard Wizard::operator = (const Wizard &wizard)
{
    delete [] spells;
    name = wizard.name;
    spellsIndex = wizard.spellsIndex;
    spells = new string[wizard.spellsIndex];
    for (int i=0; i < spellsIndex; i++)
        spells[i] = wizard.spells[i];
    return *this;
}

Destroy existing spells array so that a new one can be created that is similar to the object being copied

Middle is similar to copy constructor

We return an object of Wizard
this pointer

- *this* pointer is a special built-in pointer that is available to class’s member functions
- Points to instance of class making the function call
- Passed as hidden parameter to all non-static member functions
Operator Overloading

• Cannot change the number of operands taken by an operator
• E.g. = operator can only be a binary operator
• You can overload most operators in C++
• However, you cannot overload the following operators:
  o ?: (ternary or conditional operator)
  o . (member access or dot operator)
  o .* (pointer to member operator)
  o :: (scope resolution operator)
  o sizeof (object size operator)
Mathematical Operator Overloading

- Similar to = overloading, but with the following differences:
  - Does not return *this
  - Return type is not constant
class Wizard
{
    Wizard operator + (const Wizard &wizard);
};

Wizard Wizard::operator + (const Wizard &wizard)
{
    Wizard wiz;
    for (int i=0; i < spellsIndex; i++)
        wiz.addSpell(spells[i]);
    for (int j=0; j < wizard.spellsIndex; j++)
        wiz.addSpell(wizard.spells[j]);
    return wiz;
}