Overloading Operators

August 30, 2017

Brian A. Malloy

Overview
Copy Assignment
Output Operator

1. Overview

- Learning to overload operators is essential
- If a class has pointer data it should be in canonical form; i.e., user defined copy constructor, copy assignment, and destructor.
- We will overload operators for string:
  - copy assignment
  - output
  - string concatenation
  - non-const bracket
  - const bracket

1.1. Signatures for string Operations

```
1  class string {
2  public:
3    string();
4    string(const char*);
5    string(const string&);
6    ~string();
7    string& operator=(const string&);
8    string operator+(const string&);
9    char& operator[](int index);
10   const char& operator[] const (int index);
11 private:
12   char *buf;
13 };
14 ostream& operator<<(ostream&, const string&);
15 string operator+(const char*, const string&);
```

1.2. Overview (cont)

- Almost all operators can be overloaded
- Operators are binary or unary
- Have the same precedence as their compiler counterpart
- Can be members or friends
- Usually overloaded output operator should not be a member of a user defined class

1.3. An overloaded binary operator:

- Can be written in math form:

```
a = b;
c = a + b;
cout << stu;
```

- Or can be written in the usual form of object.function_name(params):

```
a.operator=(b)
c.operator=(a.operator+(b));
cout.operator<<(stu)
```

- Most prefer the math form

2. Copy Assignment

```
1  string& operator=(const string& rhs) {
2    if ( this == &rhs ) return *this;
3    delete [] buf;
4    buf = new char[strlen(rhs.buf)+1];
5    strcpy(buf, rhs.buf);
6    return *this;
7  }
```

- Return type is string& to permit a = b = c
- Line 2 checks for assignment to self; note that we cannot do this with *this == rhs
- On line 3 we delete the old memory
- On line 4 we allocate for rhs.buf
- On line 6 we return a reference to the current object to permit a = b = c.

Overview
Copy Assignment
Output Operator

2.1. Formula for overloading assignment:

- Check for equality of lhs & rhs
- delete storage for lhs
- Create new storage for lhs, thats size of rhs
- Copy rhs stuff to lhs
- Meyers, Item 16: "Assign to all data members in operator="
- return *this

3. Output Operator

```
1  class string {
2  public:
3    string(const char* b) :
4      buf(new char[strlen(b)+1]) {
5      strcpy(buf, b);
6    }
7    ~string() { delete [] buf; }
8    const char* getBuf() const { return buf; }
9  private:
10   char* buf;
11 };
12
13 std::ostream&
14 operator<<(std::ostream& o, const string& s) {
15   return o << s.getBuf();
16 }
```

3.1. Explanation of Output Operator

- It's a global function, the usual call is:
  ```
  string s;
  operator<<(std::cout, s);
  ```
- However, using syntactic sugar, the C++ compiler allows us to call output:
  ```
  string s;
  std::cout << s;
  ```
- The 2nd parameter to output is const string&, ⇒ getBuf() must be const member.
- operator<< is left associative; thus, we return ostream& to permit:
  std::cout << a << b;

3.2. Why not a member function?

```
1  class string {
2  public:
3    string(const char* b) :
4      buf(new char[strlen(b)+1]) {
5      strcpy(buf, b);
6    }
7    ~string() { delete [] buf; }
8    std::ostream& operator<<(std::ostream& out) {
9      return out << buf;
10   }
11 private:
12   char* buf;
13 };
14 int main() {
15   string a("dog");
16   a << std::cout; // this is backwards!
17 }
```