Overloading Operators

August 31, 2017

Brian A. Malloy

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.
- If a class has no pointer data, no need to write copy constructor or copy assignment.
- We will overload operators for string:
  - copy assignment
  - output
  - string concatenation
  - non-const bracket
  - const bracket

Overview
Copy Assignment
Output Operator
Concatenation
◀◀ ▶▶
◀ ▶
Slide 2 of 11
Go Back
Full Screen
Quit

1.1. Signatures for string Operations

```
class string {
public:
  string();
  string(const char*);
  string(const string&);
  ~string();
  string& operator=(const string&);
  string operator+(const string&);
  char& operator[](int index);
  const char& operator[] const (int index);
private:
  char *buf;
};
ostream& operator<<(ostream&, const string&);
string operator+(const char*, const string&);
```

Overview
Copy Assignment
Output Operator
Concatenation
◀◀ ▶▶
◀ ▶
Slide 3 of 11
Go Back
Full Screen
Quit

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

Overview
Copy Assignment
Output Operator
Concatenation
◀◀ ▶▶
◀ ▶
Slide 4 of 11
Go Back
Full Screen
Quit

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

Overview
Copy Assignment
Output Operator
Concatenation
◀◀ ▶▶
◀ ▶
Slide 5 of 11
Go Back
Full Screen
Quit

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.

CLEMSON UNIVERSITY

Overview

Copy Assignment

Output Operator

Concatenation

◀◀ ▶▶

◀ ▶

Slide 6 of 11

Go Back

Full Screen

Quit

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

Overview
Copy Assignment
Output Operator
Concatenation
◀◀ ▶▶
◀ ▶
Slide 7 of 11
Go Back
Full Screen
Quit

3. Output Operator

```
class string {
public:
  string(const char* b) :
    buf(new char[strlen(b)+1]) {
    strcpy(buf, b);
  }
  ~string() { delete [] buf; }
  const char* getBuf() const { return buf; }
private:
  char* buf;
};

std::ostream&
operator<<(std::ostream& o, const string& s) {
  return o << s.getBuf();
}
```

Overview
Copy Assignment
Output Operator
Concatenation
◀◀ ▶▶
◀ ▶
Slide 8 of 11
Go Back
Full Screen
Quit

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;

Overview
Copy Assignment
Output Operator
Concatenation
◀◀ ▶▶
◀ ▶
Slide 9 of 11
Go Back
Full Screen
Quit

3.2. Why not a member function?

```
class string {
public:
  string(const char* b) :
    buf(new char[strlen(b)+1]) {
    strcpy(buf, b);
  }
  ~string() { delete [] buf; }
  std::ostream& operator<<(std::ostream& out) {
    return out << buf;
  }
private:
  char* buf;
};
int main() {
  string a(”dog”);
  a << std::cout; // this is backwards!
}
```

Overview
Copy Assignment
Output Operator
Concatenation
◀◀ ▶▶
◀ ▶
Slide 10 of 11
Go Back
Full Screen
Quit

4. Concatenation

```
class string {
public:
  string(int n = 0) : buf(new char[n+1]) {
    buf[n] = '\0';
  }
  string operator+(const string& rhs) {
    string temp(strlen(buf)+strlen(rhs.buf)+1);
    strcpy(temp.buf, buf);
    strcat(temp.buf, rhs.buf);
    return temp;
  }
private:
  char* buf;
};
int main() {
  string a("cat"), b("alog");
  std::cout << a+b << std::endl;
}
```

Overview
Copy Assignment
Output Operator
Concatenation
Slide 11 of 11
Go Back
Full Screen
Quit