

Lecture Topics

- pitfalls of overloading & conversions (continued)
- overloading miscellany
- variable declarations
- extensibility: philosophy vs. reality
- profiling and tuning (more next time)

Administrivia

- Lab #1 almost ready!

Stealing a Call

- I did have to jump through more hoops than I expected to create this example.
- Of course, gcc seemed to be identifying more things as “ambiguous” than suggested in the (dated) book we’re using or in the (up-to-date?) docs on the IBM web pages.

```
class ALPHA {};  
class BETA {  
    public:  
        operator int () {return 42;}  
        friend check (const BETA& obj, int num);  
}  
class GAMMA : public ALPHA, public BETA {};
```

- consider a call of the form...

```
GAMMA g, h;  
check (g, h);
```

- Such calls go to the unique function `check` defined in BETA.
- What happens if I add the following to class ALPHA?

```
friend check (const ALPHA& obj, const ALPHA& num);
```
- answer: calls are silently transferred to the new function

- Here's a less contrived example that illustrates the potential danger of using "convenient" implicit conversions.

```
class BETA {  
    public:  
        operator int () {return 42;}  
}
```

- Once you've created the implicit conversion, it's possible to pass a BETA to any function that takes an `int`.
- So people write some code with the implicit conversion.
- Now someone else comes along and decides to create a function that takes a BETA and happens to have the same name as a function that takes an `int` in place of the BETA
- result
 - new function *probably* needs to be friend of class
 - calls are stolen
- still a little contrived
 - single argument functions are likely to be member functions, which won't have this problem
 - more arguments are less likely to match exactly by chance

Overloading Miscellany

- consider overloading array syntax (`operator []`)
- Did you think of overloading reads, writes, or both?
 - `x[i] = x[j];`
 - left side is an L-value
 - right side is some data type stored in X at index j
- implementation
 - right side probably pretty easy (look up and return)
 - if X is a complicated data structure, left side may be slower/harder
 - Can you define one function (`operator []`) that works?
 - not really
 - should there be two versions of `operator []`?
 - or find a workaround?
- example workaround (see Str. Sec. 3.7.1)
 - use an extra data structure to hack it
 - given class ALPHA that stores objects of class BETA
 - create helper class ALPHA_REF containing ALPHA* and integer
 - `operator[]` returns new ALPHA_REF
 - ALPHA_REF has two operators
 - cast operator to BETA (do the actual lookup)
 - assignment operator from BETA (do an insertion)
 - now `X1[i] = X2[j]` becomes...

```
x1.operator[] (i).operator=(x2.operator[] (j).operator BETA ( ))
```

- not all operators can be overloaded
 - member access (“.”)
 - pointer to member function invocation (“.*”)
 - conditional expressions (?:)
 - scope identification (::)
- overloading can break C’s duality
 - pointer-like objects and array-like objects not necessarily equal
 - pointer vs. array
 - `array[10]`
 - `*(array+10)`
 - pointer dereference
 - `inst->member`
 - `(*inst).member`
 - `inst[0].member`
 - not possible to change definitions equivalently because “.” can’t be overloaded
- copying vs. constructing
 - What’s the difference between the two assignments below?

```
ALPHA a;  
ALPHA b = a; // copy constructor  
b = a;      // assignment
```
 - declaration has no “old version”
 - may need work to destroy previous version
 - e.g., rehash instance in a lookup table
 - these two are **NOT** equivalent in C++
 - default version is memberwise copy for both
 - overriding one does **NOT** catch the other (other version will use default copy)
 - compiler will **NOT** warn you

Variable Declarations and Single-Assignment

- objectives
 - clean up minor scoping issues
 - e.g., loop variables, test conditions
 - scope inside loop / inside then/else code blocks
 - provide single-assignment style choice that is easier to optimize
 - avoid “empty” constructor for uninitialized objects
 - avoid need to optimize away reads of dead initialization
- extension (to C)
 - support more flexible variable declarations
 - interleaved with statements
 - note: doesn't really solve the problem
 - `if (...) {var = ...;} else {var = ...;}`
 - still need outside (dead) constructor
 - can obscure type information (buried somewhere in code)
- also encourage use of op-assign operators (`+=`, `-=`, `*=`, etc.)
 - optimizer in theory may be able to transform
 - in practice, has to worry about aliasing
 - `A = A + B;`
 - Can I update A safely before calculating sum?
 - not clear
 - if compiler has both functions, can analyze interaction
 - can't insert call to `op_and_assign(A, B, A);`
 - in contrast
 - `A += B`
 - implementation obvious expects aliasing

Philosophy vs. Reality

- “...no type...can have operations added after its definition is complete.” (Str, p. 81)
- functions valid on a class instance
 - defined at compile time (as with Simula)
 - in contrast with Smalltalk
 - which allows use of dynamic extensions, e.g., `base_pointer->new_func_name ();`
 - can't rule out questionable uses at compile time
 - thus implies run-time checking
- extensibility
 - consider base class with some non-virtual functions
 - attractive for performance reasons
 - Can I create a derived class that overrides a non-virtual function in the code of the base class?
 - No: I have to change the base class definition (add virtual keyword).
 - Java provides “final” to allow optimization
 - same issue!
 - to make an extension
 - I have to go in and remove “final.” Not much different.
- implications of definition may be subtle
 - “operations” may be implicit in the definition
 - changing the definition may
 - implicitly change the meaning of code using the definition
 - without producing any errors or warnings
 - implicit conversions
 - single-argument constructors
 - cast operators

Profiling and Tuning

- sub-topics
 - measuring operation timing
 - sample-based profiling
 - capturing more details (oprofile)
 - architectural hazards (profileme)

Measuring Operation Timing

- scenario
 - you've written a library or module (e.g., NetLink)
 - how do you do “unit tuning”?
 - i.e., optimization without major driver apps
- be careful
 - avoid premature optimization
 - you need to know how library will be used before you implement it
 - with NetLink, networking and system call costs dominate library costs (as I mentioned, but we'll confirm)
- brainstorming opportunity
 - given a short operation sequence (maybe even one)
 - what makes it hard to measure accurately?