Programming Languages and Tools: Programming with C++ CS:3210:0003

Lecture/Lab #18

Operator Overloading

- C++ implements operators as functions
- Operator overloading: overload operators to work with different data types
- Rules:
  1. C++ only allows you to overload existing operators
  2. At least one of the operands in the overloaded operator must be a user-defined type (standard library classes are user-defined)
  3. The arity, precedence, and associativity of the operator cannot be changed
- Best Practices:
  1. Keep function of overloaded operator close to the original function
  2. If meaning of overloaded operator isn't clear, use a named function
  3. If operator doesn't modify operands (ex: +), return result by value
  4. If operator modifies operand (ex: ++), return by reference

Friends

- Private members of a class can only be accessed by member functions or friends of class
  - A friend non-member function has access to private/protected members of class
  - A friend class has access to private/protected members of class
  - Instead of giving a whole class access, we can give it to a friend member function of an external class
- Friendship must be granted by class by declaring function preceded by `friend`
- Friend non-member function can also be defined inside class (but remains a non-member)

Operator Overloading

- Ways to overload:
  1. Using friend functions: If the overloaded operator needs to access private members of class, it can be declared as a friend

Activity

From `03-27/02eqfrac.cpp`,

1. Replace `mulFrac` with an overloaded * operator
2. Replace `eqFrac` with an overloaded == operator

Operator Overloading

- Ways to overload:
  1. Using friend functions: If the overloaded operator needs to access private members of class, it can be declared as a friend
  2. Using normal functions and getters: Getter functions can give a normal function access to private members so an overloaded operator can use getters

- Prefer non-friend functions to friend functions if getters already exist
- When overloading insertion/extraction (>>/<<), the overloaded function must return by reference since `std::ostream` and `std::istream` disallow being copied

Activity

Overload `<<` and `>>` so we can input and output fractions directly

1. Output fractions `m/n` (`cout` is an object of the `ostream` class)
2. Input fractions by first getting the numerator, then the denominator (`cin` is an object of the `istream` class)

Operator Overloading

- Ways to overload:
  1. Using friend functions: If the overloaded operator needs to access private members of class, it can be declared as a friend
  2. Using normal functions and getters: Getter functions can give a normal function access to private members so an overloaded operator can use getters
  3. Using member functions: overloaded operator becomes member of left operand and left operand becomes implicit (via `this` pointer)
- Prefer non-friend functions to friend functions if getters already exist
- When overloading insertion/extraction (`>>`/`<<`), the overloaded function must return by reference since `std::ostream` and `std::istream` disallow being copied

How to Overload?

- Assignment (=), subscript ([]), function call(()), member selection (->) must be overloaded as member functions, it's required by C++
- As a member function, overloaded operator must be added as a member of the left operand
- If the operator modifies an operand, overload as member (if possible)
  - When not possible, overload as non-member function (possibly friend)
  - Ex: `istream` and `ostream` classes can't be modified
- If possible (if getters already exist), prefer non-friend

Return by Reference

- Return a reference bound to the object being returned
- Avoids making a copy of return value
- Ex:

```
int& add (int x, int y);
```

- Objects returned by reference must live beyond the scope of the function returning
  - Otherwise, dangling reference
- A static variable outlives the function it was defined in