CSE 143

Overloading

[Chapter 8, pp. 377-381]

Name Scope and Visibility

- Variable and function names may be repeated in different scopes

```
int myVar;
void Snork(int myVar)
{
   ...
   while ( a > b){
     int myVar = 0;
     ...
   }
}
```

- Style issue: using same name in multiple scopes can be confusing

Overloading

- Reusing names
  - Function names can be reused
  - Differentiate based on scope and context
    `Global vs. class vs. local, etc.`
- Different functions in **same** scope can have same name if argument number and/or types different
  - Name is said to be overloaded
- Operators (+, =, etc.) can also be overloaded (ex. int+int, double+double, Complex+Complex)

Overloaded Function Names

- Useful if one name has multiple meanings or multiple interfaces
- Constructors are common example of overloading since all must have same name
- Compiler determines which function to call at compile-time

Resolving Overloaded Functions

To "resolve" mean to decide which version of the overloaded function is being called

- Determined by matching actual arguments against possible formal arguments
- Compiler gives error if not exactly one matches
- Complete matching algorithm rather complex
- If match is not exact, automatic type conversions are used

Matching Algorithm

- Function declarations

```
void Snark(int);
void Snark(double);
void Snipe(char []);
void Snipe(double);
void Sneep(char);
void Sneep(double);


Snark(1);        // Integer Snark
Snipe(1);        // Double Snipe
Sneep(1);        // Ambiguous
```

Example of Resolving

- Function declarations

```
void PrintData(int data) {
 cout << "int = " << data << endl; }
void PrintData(char data) {
  cout << "char = '" << data << "'\n"; }
void PrintData(double data) {
  cout << "double = " << data << endl; }
```

- Which calls are valid?

```
PrintData(3);
PrintData("Hello");
PrintData(3.14159);
PrintData('m');
```


Overloaded Operators

- For convenience, can define functions named +, -, *, =, /, ==, etc. on classes
  - Gives natural expression to some operations
  - Very confusing if abused
- Operator functions may be members or friends
  - Function "names" are **`operator+`**, **`operator-`**, ...
  - Several ways of calling them

+ as Friend Function

- Declaration:

```
class Complex {
 ...
 friend Complex operator+(Complex,Complex);
 ...
};
```

- Implementation:

```
Complex operator+(Complex a, Complex b){
  return Complex(a.re+b.re, a.im+b.im);
}
```

- Use: **`c = d+e;`**
- Means: **`c = operator+(d,e);`**
- Critique?

+ as Member Function

- Declaration:

```
class Complex {
 ...
 Complex operator+(Complex);
 ...
};
```

- Implementation:

```
Complex Complex::operator+(Complex z){
  return Complex(re+z.re, im+z.im);
}
```

- Use: **`c = d+e;`**
- Means: **`c = d.operator+(e);`**
- Critique?