C++: Functions, Methods, Operators

Announcement

- RIT Career Fair
  - Thursday, Oct 3rd 1pm – 7pm
  - Friday, Oct 4th 9am – 5pm (interviews)
- Clark Gym
- www.rit.edu/co-op/careers

Announcement

- Date for Exam 1 will probably be changed
  - New date TBD.
  - Will know new date by Thursday.

Project

- Questions?
- Yes, there will be only one project.
  - Typo on Web page.
- Everyone have a partner?
- Please e-mail me with the name of your partner and I will assign you a group account.

Speaking of e-mail

- A test e-mail was sent to the class.
- If you did not receive this test message, please check your e-mail address listed in LDAP.

Plan for this week

- Today: Functions, Methods, Operators
  - Overloading Functions
  - Overloading Operators
  - Assignment Operator
- Tomorrow: Constructors, Destructors
  - Added bonus: enumerated types, assertions
- Thursday: Start of memory management.

Function overloading

- The same function / method name can be used several times:
  - The argument set and return type must be different for each function definition
  - Overloaded functions cannot differ by return type alone.

Function overloading

```
class Foo {
public:
      char f();
      char f (int x);
      char f (int x, int y);
      double f (double x, double y);
      int f (int x);   // not allowed
}
```

Function overloading

- Why bother?

```
void print_int (int i);
void print_char (char c);
void print_double (double d);
```

Compared to

```
void print (int i);
void print (char c);
void print (double d);
```

Operator overloading

- All C++ operators can be overloaded on a class by class basis.
- Overloaded operators call specially named class methods.
  - Keyword `operator` followed by operator to be overloaded.
  - E.g
    - `operator+`

Operator overloading

```
class Complex
{
private:
    double re, im;
public:
   Complex (double r, double i);
   Complex operator+( Complex &c ) const;
   Complex operator-() const;
   bool operator== (Complex &c) const;
   Complex& operator+= (Complex &c) ;
   Complex& operator+= (double d);
};
```

Operator overloading

- Once overloaded, operators can be used in the same manner as for basic types.
- E.g.

```
Complex c1, c2, c3;
double d=5.0;

c2 = c1 + c3;
c3 = -c1;
c3+=c2;
c3+=d;
if (c3 == c1) { … }
```

Operator overloading

- Using overloaded operators is just a shorthand for calling the specially named class methods.
- E.g.

```
c2 = c1 + c3;
```

  - Is the same as

```
c2 = c1.operator+ (c3);
```

Operator overloading

- What would the definition of an overloaded operator function look like?

```
Complex Complex::operator+( Complex &c ) const
{
  return Complex (re + c.re, im + c.im);
}

bool Complex::operator== (Complex &c) const
{
    return ((re == c.re) && (im == c.im));
}
```

Operator overloading

- What would the definition of an overloaded operator function look like?

```
Complex& Complex::operator+= (Complex &c)
{
  re += c.re;
  im += c.im;
  return (*this);
}
```

Operator overloading

- It would be nice for all operators to return references…but this is difficult

```
Complex& Complex::operator+( Complex &c ) const
{
   // Here memory associated with CC will go
   // away once function completes
   Complex CC(re + c.re, im + c.im);
   return (CC);  // Returning ref to a var
                 // that will no longer exist.
}
```

Operator overloading

- It would be nice for all operators to return references…but this is difficult

```
Complex& Complex::operator+( Complex &c ) const
{
   // We could try to allocate on the free store
   Complex *CC
     (new Complex (re + c.re, im + c.im));
   return (*CC);  // but who's going to clean this
                  // up?
}
```

Operator overloading

- So when can references be returned?
  - Operators that modify themselves and return references to themselves.
  - Const operators, which just use the values of an object will generally create an pass back a new object.
  - Logical operators should return `bool`.

Operator overloading

- Overloaded operators can also be defined globally as non-members (outside of the class definition)

Operator overloading

- Friends
  - By declaring a function as a `friend`, we allow it access to a class's private data members (both data and methods)

Operator overloading

- Global operator definitions

```
friend Complex operator+( Complex &c1, Complex &c2 );
friend Complex operator-(Complex &c1);
friend bool operator== (const Complex &c1, const
  Complex &c2);
friend Complex& operator+= (Complex &c1, const Complex
  &c2);
friend Complex& operator+= (Complex &c, double d);
```

Operator overloading

- Why use friend?
  - Used for operators that have **another class** as the left operand
    - E.g. << (as we'll see in next slide)
  - permit operators to be **commutative**.

```
Complex c1, c2;
double d;
c1 = c2 + d;
c1 = d + c2;  // Not allowed if member
```

Operator overloading

- Global friend operators can be declared anywhere.
  - `public` and `private` don't apply to them.

```
class Complex
{
private:
    double re, im;
    friend Complex operator+( Complex &c1, Complex &c2 );
    friend Complex operator-(Complex &c1);
public:
    Complex (double r, double i);
    friend bool operator== (const Complex &c1, const Complex
   &c2);
    friend Complex& operator+= (Complex &c1, const Complex &c2);
    friend Complex& operator+= (Complex &c, double d);

};
```

I/O overloaded operators

- Overloading << and >>

```
friend ostream& operator<<(ostream& output,
  const Complex c) {
  output << c.re << " + " << c.im << " I"
  return output;
}

Complex c1 (1.0, 2.0);
cout << "My complex number is: " << c1;

My complex number is: 1.0 + 2.0 i
```

Overloading operators

- Questions?

Assignment operator

- operator=
  - Called when an assignment is made
  - Copies all relevant data from object assigner to assignee.
  - Must be declared as a class member
  - Should check for self-assignment!

Assignment operator

```
class Complex
{
private:
    double re, im;
public:
    Complex & operator= (Complex &c);
    ...
}

Complex c1, c2;
c2 = c1;   // is the same as saying
c2.operator= (c1);
```

Assignment operator

```
Complex & Complex::operator= (Complex &c)
{
  if (c != (*this)) {
     re = c.re;
     im = c.im;
  }
  return (*this);
}
```

Assignment operator

- Note that the assignment operator returns a reference to itself
  - This is to allow statements like:

```
Complex c1, c2, c3;
c3 = c2 = c1;
```

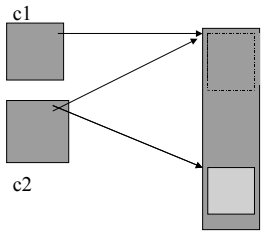
Assignment operator

- If no assignment operator is defined for a class, the default assignment operator is used.
  - Member by member copy of data from one object to another.
  - Can be troublesome if class have pointers as data members.

Assignment operator

```
class Foo
{
private:
    int
  *array_member;
    int asize;
    ...
}

Foo c1, c2;
c1 = c2;
delete c1;
delete c2;
```



Free store

Assignment operator

```
Foo & Foo::operator= (Foo &F)
{
   // cleanup old array
  delete array_member;

  // allocate new array
  asize = F.asize;
  array_member = new int (F.asize);

  // copy
  for (int i=0; i<asize; i++)
      array_member[i] = F.array_member[i];
}
```

Assignment operator

Questions?

Summary

- Functions
  - They can be overloaded
- Operators
- Assignment Operator

Next time

- Constructors
- Destructors
- Enumerated Types
- Assertions