# Function and Operator Overloading

Introduction to Programming

12/31/2015

## Function Overloading

- C++ allows functions to have the same names
  - But the parameters, number or types, need to be different
  - Function overloading
  - To have similar functions with different data types
  - Return type is irrelevant
- C++ compiler will use the best matched function
  - Argument promotion might be performed
- Example: exq1.cpp

## Operator Overloading

- Operators: shorthand representations help in technical and nontechnical communications
- C++ provides user-defined types, classes, and most of the operators are not defined for these classes
- Operators for classes are user-defined
- For example, `+` and `*` are well known for complex numbers
- Define these operators can help program development
- Many of the most obvious uses of operator overloading are for concrete types.
- But the usefulness of user-defined operator is not restricted to concrete types
- Example: exq20.h, exq21.cpp, exq2.cpp

## Operator Functions

- Most of the operators can be defined for classes
  `+` `-` `*` `/` `%` `^` `&` `|` `~` `!` `=` `<` `>` `+=` `-=` `*=` `/=` `%=` `^=`
  `&=` `|=` `<<` `>>` `>>=` `<<=` `==` `!=` `<=` `>=` `&&` `||` `++` `--` `->*`
  `,` `->` `[]` `()` `new` `new[]` `delete` `delete[]`
- The following operators can not be redefined
  - `::` scope resolution
  - `.` member selection
  - `.*` member selection through pointer to member
  - `? :` ternary condition expression
  - `sizeof`
  - `typeid`
- The first 3 operators take a name, rather than value, as the second operand

## Operator Functions

- It is not possible to define new operator token
  - For example: `**` is not defined
- The name of an operator function is the keyword `operator` followed by the operator itself
  - For example: `operator+`
- Two ways of using operator function
  - Shorthand:

```
a+b
```

  - Explicit function call:

```
a.operator+ (b); // a function call
```

## Binary Operators

- A binary operator can be defined by either a nonstatic member function taking one argument or a nonmember function taking two arguments.
  - Example:

```
class myClass {
myClass operator+(myClass b);    //nonstatic function member
};
myClass operator-(myClass a, myClass b) // utility function
```

  - Usage:

```
c=a+b;
c=a.operator+(b);
c=operator-(a,b);
```

## Unary Operators

- A unary operator, whether prefix or postfix, can be defined by either a nonstatic member function taking no argument or a nonmember function taking one argument.
- Example:

```
class myClass {
myClass operator++(); //nonstatic member function
}
myClass operator--(myClass a); // utility function
```

- Usage:

```
++a;
a.operator++();
operator++(a);
```

## Postfix Unary Operator

- For any postfix unary operator `a++`, it can be interpreted as either `a.operator++(int)` or `operator++(a,int)`
- Example:

```
class myClass {
myClass operator++(int); //nonstatic member function
}
myClass operator--(myClass a,int n); // utility function
```

- Usage:

```
a++;
a.operator++(1);
operator++(a,1);
```

- Please note the differences in implementation for prefix and postfix unary operators

## Binary and Unary Operator Examples

```
class X {
  X* operator&();      // prefix unary operator &
  X operator&(X);      // binary &
  X operator++(int);   // postfix increment
  X operator&(X,X);    // error ternary
  X operator/();       // error unary
};

X operator-(X);        // prefix unary minus
X operator-(X,X);      // binary minus
X operator--(X&,int);  // postfix decrement
X operator-();         // error no argument
X operator-(X,X,X);    // error, ternary
X operator%(X);        // error, unary
```

## Operators and User-Defined Types

- An operator function must be a member function or takes at least one user-defined type argument
  - Thus not changing existing expression (without user-defined objects)
- Operator function with a basic type as the first argument cannot be a member function
- $\boxed{=}$, $\boxed{[]}$, $\boxed{()}$, $\boxed{->}$ must be nonstatic member function so that the first operand is an `lvalue`.
- Combinations of operators are not assumed
  - $\boxed{+=}$ is not $\boxed{+}$ and $\boxed{=}$
  - $\boxed{++}$ is not $\boxed{+1}$ or $\boxed{+=\ 1}$
- $\boxed{=}$(assignment), $\boxed{\&}$(address of) and $\boxed{,}$(sequencing) are predefined
  - but can be made to be private and thus not available to general users

## Operators in Namespaces

- Operator function can be defined in namespaces
- Operator function is resolved for `X@Y` by
  - If `X` is a class, look for $\boxed{\texttt{@}}$ as member function of `X` or the base of `X`
  - Look for declarations of `opertor@` in the context surrounding `X@Y`
  - If `X` is defined in namespace `N`, look for $\boxed{\texttt{@}}$ in `N`
  - If `Y` is defined in namespace `M`, look for $\boxed{\texttt{@}}$ in `M`
- Unary operator is resolved analogously

## Complex Number Type

- Operators can be defined such that most math shorthand symbols can be applied directly
  - Need copy assignment, assign with scalar, addition with scalar, adding to scalar, unary $-$, multiplication, etc
  - Minimize the number of functions that directly manipulate the representation of an object
    - Keep as member function
  - Other operators defined as nonmember functions
- Example: exq30.h, exq31.cpp, exq3.cpp

## Friends

- Member functions specify 3 things
  - They can access private data members
  - Function is in the scope of the class
  - The function must be invoked through an object of the class
- Static member function has only the first two properties
- A friend function has only the first property
- A friend function can be declared in either private or public part
- A member function can be a friend of another class
- All the member functions can be Friends of another. Shorthand representation

```
class C1 {          // all member functions of C2
  friend class C2; //      are friends of C1
};
```

- Choose between making a class a member (nested class) or nonmember friend

## Subscripting

- The `operator[]` function can be used to give subscripts a meaning for a class object.
- The second argument, the subscript, may be of any type
- It can be used to define vectors, associative arrays, etc
- Example: exq4.cpp

```
i=tw["新竹"];
```

## Constructors and Destructors

- A constructor is called when an object is created
- Three types of constructors exq5.cpp
  - Constructor without initialization

```
Complex z1;
```

  - Initialization constructor

```
Complex One(1,0);
```

  - Copy constructor

```
Complex z2=One;
```

- In function call, the constructor is called for
  - Arguments passed by value
  - And may be for return value
- A destructor is called when a variable is no longer needed
  - A variable is going out of scope
    - End-of-block for local variables
    - End-of-function-call for Passing-by-value arguments

## Summary

- Function overloading
- Operator overloading
- Operator functions
  - Binary operators
  - Unary operators
  - Postfix unary operators
  - Operator and user-defined types
- Complex numbers
- Subscripting