Operator Overloading

Humayun Kabir
Professor, CS, Vancouver Island University, BC, Canada

Operator Overloading

- You can redefine or **overload** the behaviour of most built-in **operators** in C++ on a class.
- You **can overload** any of the following operators:
  - Arithmetic: **+ - * / % ++ --**
  - Bitwise: **^ & | ~ ! = << >>**
  - Assignment: **= += -= *= /= %= ^= &= |= >>= <<=**
  - Logical: **== != <= >= && ||**
  - Others: **, ->* -> ( ) [ ] new delete new[] delete[]**
- You cannot overload any of the following operators:

  :: (scope resolution)

  . (member access)

  .* (member access through pointer to member)

  ?: (ternary conditional)

Operator Overloading

- You cannot create any new operators, such as **** <> &|**
- You cannot change the **precedence**, **grouping**, or **number of operands** of operators.
- At **least one** of the **operands** of your **overloaded operator** must be of **your class type**.
- An overloaded operator is called an **operator function**.
- You declare an **operator function** with the keyword **operator** preceding the **operator symbol**.
- **Keyword** operator and an operator **symbol** together becomes the **name** of a operator function.

Operator Overloading

- Operator functions can be either **member functions** or **friend fucntions**.
- **Unary operators** must be overloaded as **member functions**.
- **Binary operators** with **both operands** of **your class type** should be overloaded as **member functions**, although they can be overloaded as friend functions.
- **Binary operators** with the **first operand** of **different type** must be overloaded as **friend functions**.

Member Operator Function

```
class Point {
        private:
                        int x;
                        int y;
        public:
                        Point(int x, int y): x(x), y(y) {}
                        Point& operator ++ () {
                             ++x;
                             ++y;
                             return *this;
                        }
};
int main() {
        Point p1(1,2);
        ++p1;       // p1.operator++(),  p1(2,3)
        return 0;
}
```

Member Operator Function

```
class Point {
        private:
                        int x;
                        int y;
        public:
                       Point(int x, int y): x(x), y(y) {}
                       Point operator + (const Point &rhs) {
                            return Point(x+rhs.x, y+rhs.y);
                       }
};
int main() {
         Point p1(1,2);
         Point p2(3,4);
         Point p3 = p1 + p2;    // p1.operator+(p2),  p3(4,6)
         return 0;
}
```

Friend Operator Function

```
class Point {
        private:
                        int x;
                        int y;
        public:
                        Point(int x, int y): x(x), y(y) {}
                        friend Point operator + (const Point &lhs, const Point &rhs);
};
Point operator + (const Point &lhs, const Point &rhs); {
            return Point(lhs.x+rhs.x, lhs.y+rhs.y);
}
int main() {
        Point p1(1,2);
        Point p2(3,4);
        Point p3 = p1 + p2;    // operator+(p1, p2),  p3(4,6)
        return 0;
}
```

Friend Operator Function

```
class Point {
        private:
                        int x;
                        int y;
        public:
                       Point(int x, int y): x(x), y(y) {}
                       friend Point operator + (const int z, const Point &rhs);
};
Point operator + (const int z, const Point &rhs); {
            return Point(z+rhs.x, z+rhs.y);
}
int main() {
         Point p1(1,2);
         Point p2 = 5 + p2;      // operator+(5, p2),  p3(6,7)
         return 0;
}
```

Friend Operator Function

```
class Point {
        private:
                int x;
                int y;
        public:
                Point(int x, int y): x(x), y(y) {}
               friend std::ostream& operator << (std::ostream& out, const Point &rhs);
};
std::ostream& operator << (std::ostream& out, const Point &rhs); {
                out<<"<"<<rhs.x<<","<<rhs.y<<">";
                return out;
}
int main() {
                Point p1(1,2);
                std::cout<<p1<<std::endl;        // operator<<(std::cout, p1),  <1,2>
                return 0;
}
```