Operator Overload

Operator function

- When at least on operand is an object, an operator is a function
- Operator overload: define an operator function

```
int operator + (int x, int y) {
    return x + y;
}
```

```
8.cpp:7:5: error: overloaded 'operator+' must have at least one parameter of
      class or enumeration type
int operator + (int x, int y) {
    ^
1 error generated.
```

Operator Overload

- As a global function

```
class A {};

A operator + (int x, A y) {
    cout << "In operator +" << endl;
    return A();
}

int main() {
    A x;
    x = 100 + x;
    x = operator+(100, x);
}
```

Operator Overload

- C++中为每个operator规定了参数的个数，但没有规定参数或返回值的类型

```
A operator - (int x, A y, A z) {
    return A();
}
```

```
8.cpp:25:3: error: overloaded 'operator-' must be a unary or binary operator
      (has 3 parameters)
A operator - (int x, A y, A z) {
  ^
```

Operator Overload

- 定义操作符函数的时候需要注意该操作符本身的含义与函数的功能是否相关
- 并注意操作符的参数和返回值的类型与用户的习惯认识是否一致
- 一般情况下不必使用操作符: 因为函数具有名字能更容易理解其意义，除非是有明确和常用的数学定义，且该函数很常用以至于能明显提高编程效率
- 实际开发中非常常见：四则运算符（+、-、*、/、+=、-=、*=、/=）和关系运算符（>、<、<=、>=、==、!=）都是数学运算符

```
// bool operator == (A x, A y) {}
A operator == (int x, A y) {
    return x + y;
}
A equals(int x, A y) {
    return x + y;
}
```

这样的操作符，类似与下面的函数，可以认为是逻辑错误：给错了名字

Operator Overload

- Not all operators can be overloaded
  - 能够重载的运算符包括
    - + - * / % ^ & | ~ ! = < > += -= *= /= %= ^= &= |= << >> <<= >>= == != <= >= && || ++ -- , ->* -> () [] new new[] delete delete[]
  - 不能被重载的运算符包括
    - 长度运算符sizeof、条件运算符: ?、成员选择符.和域解析运算符::

Overload []

- [] 必须以成员函数的形式进行重载
- 有时需要提供2个 []

```
int main() {
    Matrix m(3,4);
    const Matrix & r = m;
    cin >> m[2.3];
    cout << r[2.3] << endl;
}
```

```
class Matrix
{
    int rows, columns;
    vector<double> data;
public:
    Matrix(int rows, int columns) : rows(rows), columns(columns) {
        data.resize(rows * columns);
    }
    double & operator[](double i) {
        return data[int(i)*columns + 10*int(i-int(i))];
    }
    const double & operator[](double i) const {
        return data[int(i)*columns + 10*int(i-int(i))];
    }
};
```

例子中使用一个double
提供两个index。
这并不是一个好的做法

Overload []

- 当缺少第1个[]时

```
class Matrix
{
    int rows, columns;
    vector<double> data;
public:
    Matrix(int rows, int columns) : rows(rows), columns(columns) {
        data.resize(rows * columns);
    }
    //·double·&·operator[](double·i)·{
    //·-return·data[int(i)*columns·+·10*int(i-int(i))];
    //·}
    const double & operator[](double i) const {
        return data[int(i)*columns + 10*int(i-int(i))];
    }
};
```

```
int main() {
    Matrix m(3,4);
    const Matrix & r = m;
    cin >> m[2.3];
    cout << r[2.3] << endl;
}
```

```
8.cpp:95:6: error: invalid operands to binary expression ('std::__1::istream'
      (aka 'basic_istream<char>') and 'const double')
        cin >> m[2.3];
        ~~~ ^  ~~~~~~
```

Overload []

- 当缺少第2个[]时

```
class Matrix
{
    int rows, columns;
    vector<double> data;
public:
    Matrix(int rows, int columns) : rows(rows), columns(columns) {
        data.resize(rows * columns);
    }
    double & operator[](double i) {
        return data[int(i)*columns + 10*int(i-int(i))];
    }
    //·const·double·&·operator[](double·i)·const·{
    //··return·data[int(i)*columns·+·10*int(i-int(i))];
    //·}
};
```

```
int main() {
    Matrix m(3,4);
    const Matrix & r = m;
    cin >> m[2.3];
    cout << r[2.3] << endl;
}
```

```
8.cpp:96:11: error: no viable overloaded operator[] for type 'const Matrix'
        cout << r[2.3] << endl;
                ~^~~~
```

Overload type operator

- 类型操作符是单目运算符
  - 只能重载为成员函数，不能重载为全局函数
  - 不能有参数或返回值
  - 提供类型转换规则

Overload type operator

```
class Matrix
{
    int rows, columns;
    vector<double> data;
public:
    Matrix(int rows, int columns) : rows(rows), columns(columns) {
        data.resize(rows * columns);
    }
    double & get(int r, int c) {
        return data[r * columns + c];
    }
    const double & get(int r, int c) const {
        return data[r * columns + c];
    }
    ...
```

Overload type operator

```
    ...
    Matrix(int columns) : rows(1), columns(columns) {
        data.resize(columns);
    }
    Matrix(const vector<double> & data) :
        rows(1), columns(data.size()), data(data) {}
    operator int() const { return data.size(); }
    operator vector<double>() const { return data; }
};

int main() {
    Matrix m(10);
    int size = m;
    cout << "size is " << size << endl;
    cout << "size is " << int(m) << endl;
    cout << "size is " << (int)m << endl;
    vector<double> data = m;
}
```

int → Matrix

vector<double> → Matrix

Matrix → int

Matrix → vector<double>

Operator Overload

- 重载为成员函数，还是全局函数？

```
    ...
    Matrix(double x) : rows(1), columns(1) {
        data.resize(1);
        data[0] = x;
    }
    Matrix & operator += (const Matrix & y) {
        *this = *this + y;
        reutrn *this;
    }
};
Matrix operator + (const Matrix & x, const Matrix & y) {
    ... // 允许左或右操作数为double
}
int main() {
    Matrix m(3,4);
    ... // 出事化m中的元素
    cout << 10 + m << endl;
    cout << m + 100 << endl;
}
```

Overload << and >>

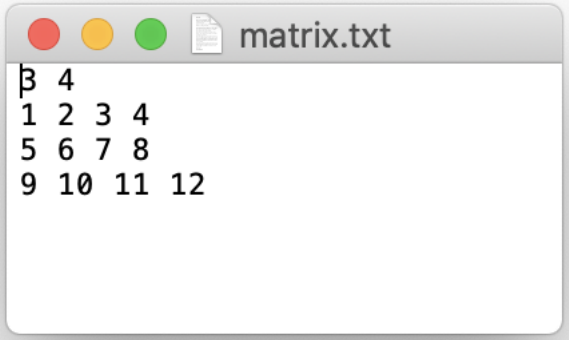
```
    ...
    int size(int dim) const {
        if (dim == 0) return rows;
        return columns;
    }
};

ostream & operator << (ostream & out, const Matrix & m) {
    out << m.size(0) << " " << m.size(1) << endl;
    for (int i = 0; i < m.size(0); ++ i) {
        for (int j = 0; j < m.size(1); ++ j)
            out << m.get(i, j) << " ";
        out << endl;
    }
    return out;
}
```

Overload << and >>

```
istream & operator >> (istream & in, Matrix & m) {
    int rows, columns;
    in >> rows >> columns;
    Matrix tmp(rows, columns);
    for (int i = 0; i < rows; ++ i)
        for (int j = 0; j < columns; ++ j)
            in >> tmp.get(i, j);
    m = tmp;
    return in;
}
```

Overload << and >>




```cpp
#include <fstream>
using namespace std;

int main() {
    Matrix m1(3,4);
    for (int i = 0; i < 3; ++ i)
        for (int j = 0; j < 4; ++ j)
            m1.get(i, j) = i * 4 + j + 1;

    ofstream out_file("matrix.txt");
    out_file << m1;
    out_file.close();

    Matrix m2(4, 5);
    ifstream in_file("matrix.txt");
    in_file >> m2;
    in_file.close();

    cout << m2;

}
```

Overload ++ and --

```
    ...
    Matrix & operator ++ () {  // ++i, 前置形式
        for (int i = 0; i < data.size(); ++ i) ++ data[i];
        return *this;
    }
    Matrix operator ++ (int) {  // i++, 后置形式
        Matrix prev_value(*this);
        for (int i = 0; i < data.size(); ++ i) ++ data[i];
        return prev_value;
    }
};
```

Overload ++ and --

```
...

int main() {
    Matrix m1(3,4);
    for (int i = 0; i < 3; ++ i)
        for (int j = 0; j < 4; ++ j)
            m1.get(i, j) = i * 4 + j + 1;
    cout << ++ m1 << endl;
    cout << m1 ++ << endl;
    cout << m1 << endl;
}
```

```
3 4
2 3 4 5
6 7 8 9
10 11 12 13

3 4
2 3 4 5
6 7 8 9
10 11 12 13

3 4
3 4 5 6
7 8 9 10
11 12 13 14
```

Overload ()

```
// double & get(int r, int c) {
//  return data[r * columns + c];
// }
// const double & get(int r, int c) const {
//  return data[r * columns + c];
// }
double & operator () (int r, int c) {
    return data[r * columns + c];
}
const double & operator () (int r, int c) const {
    return data[r * columns + c];
}
```

Overload ()

```
int main() {
    Matrix m(3,4);
    for (int i = 0; i < 3; ++ i)
        for (int j = 0; j < 4; ++ j)
            m(i, j) = i * 4 + j + 1; // #1
    const Matrix & r = m;
    for (int i = 0; i < 3; ++ i) {
        for (int j = 0; j < 4; ++ j)
            cout << r(i, j) << " "; // #2
        cout << endl;
    }
}
```

```
1  2  3  4
5  6  7  8
9  10  11  12
```

其它

- 重载不能改变运算符的优先级和结合性
- 运算符重载函数不能有默认的参数
- 大多数运算符重载函数既可以作为类的成员函数，也可以作为全局函数
  - 只能作为类的成员函数的操作符
    [] () = -> new delete new[] delete[] ...

– 必要时重载 =

- 以避免两个对象冲突地使用同一个资源