Object-Oriented Programming

Lecture 4

Agata Półrola
<agata.polrola@wmii.uni.lodz.pl>

Indexing operator [ ]

- must be as a class method (member function) with single argument
- should return a reference for use on the left-hand side of the assigment operator

```
class Vector
{
  double x, y;
 public:
  Vector(double _x = 0, double _y = 0) { x = _x; y = _y; }
  double& operator[](int i)
  {
    return (i == 1) ? x : y;
  }
};

Vector x(2, -3), y(-1, 4);
double d = x[1] + y[2]; //indexing operator calls
x[1] = 5; //also here (thanks to the reference returned)
```

Shift operators << and >>

- overloaded typically for input/output streams
- the left operand is of the other type (ostream or istream), so the operator must be a friend function and it should pass and return references to the stream (for the cascade call possibility)

```
class Vector
{
  double x, y;
 public:
  Vector(double _x = 0, double _y = 0) { x = _x; y = _y; }
  friend ostream& operator<<(ostream &stream, const Vector &v);
  friend istream& operator>>(istream &stream, Vector &v);
};
ostream& operator<<(ostream &stream, const Vector &v)
{
  stream << '[' << v.x << ',' << v.y << ']';
  return stream;
}
Vector x(2, -3), y(-1, 4) ;
cout << x << ' ' << y << endl; //shift operator cascade calls
```

Assignment operator =

- must be a class method (member function) with single argument being a reference to an existing object of the same type and it should return a reference for cascade usage
- automatically created if (and only if) no explicitly defined but copying the fields by simple bit-copy (risky when there are some pointer fields), so always overload the assignment operator for classes with pointer fields

```
class Vector
{
  double x, y;
 public:
  Vector(double _x = 0, double _y = 0) { x = _x; y = _y; }
  Vector& operator=(const Vector &v)
  {
    if (this != &v) { x = v.x; y = v.y; }
    return *this;
  }
};
Vector x(2, -3), y, z;
z = y = x; //assignment cascade calls
```

Converting operators

- with a type name after the keyword operator, without any arguments and returned value
- it must be a class method (member function)

```
class Vector
{
  double x, y;
 public:
  Vector(double _x = 0, double _y = 0) { x = _x; y = _y; }
  operator double() const
  {
    return x;
  }
};

Vector x(2, -3);
double d = x; //implicit conversion Vector → double
double e = x + d; //also here, but some ambiguity possibilities
```