연산자 오버로딩
(Chapter 8 Operator overloading ...)

숭실대학교
김강희 교수
(khkim@ssu.ac.kr)

순서

- 이론:
  - 개요
  - const 함수들
  - friend 함수들과 클래스들
  - references
  - 개요 2

연산자 오버로딩 개요

- 연산자들(+, -, %, ==, ...)을 함수로 이해하자
  - 예: 수식 x + 7 은 '+' 라는 이름의 함수, x라는 인자, 7이라는 인자로 이해할 수 있다. 즉 +(x, 7) 로 이해할 수 있다.
- 객체들을 피연산자로 삼는 연산자들을 정의할 수 있다.
  - 예: 행렬 객체 A + B

```
const Matrix operator+(const Matrix& m1, const Matrix& m2) {
        ...
        return Matrix(array, cy, cx);
}
```

    - 인자들은 reference 로 전달하고, const 키워드를 붙인다.
    - 리턴형은 Matrix 이고 const 키워드를 붙인다.
      - 생성자 함수는 anonymous object 를 리턴한다.
      - const 키워드가 없으면, 다음과 같이 비직관적인 수식을 허용하게 된다: (m1 + m2).input()
    - 위 정의는 멤버 함수로서의 정의가 아니다.

연산자 오버로딩 개요

- 객체들을 피연산자로 삼는 연산자들을 정의할 수 있다.
  - 예: 행렬 객체 A == B

    const bool operator==(const Matrix& m1, const Matrix& m2);
- 이항 연산자(binary operator) 외에 단항 연산자(unary operator: -, ++, --)도 오버로딩이 가능하다.
  - 예: A = -B

    const Matrix operator-(const Matrix& m); // 단항 연산자 오버로딩

    const Matrix operator-(const Matrix& m1, const Matrix& m2); // 이항 연산자 오버로딩
- 연산자 오버로딩을 위한 함수들을 멤버 함수로 정의하면?
  - calling object 가 하나의 피연산자가 되기 때문에, 함수 인자가 하나 줄어든다.
  - 예: 행렬 객체 A + B → A.+B 라는 표현으로 이해함

    const Matrix Matrix::operator+(const Matrix& obj);

연산자 오버로딩 개요

- 함수 호출 연산자 () 도 오버로딩이 가능하다.

  Matrix A(array, 3,3);

  A(1) // 인자 1개를 갖는 오버로딩

  A(1,2) // 인자 2개를 갖는 오버로딩

  - 함수 호출 연산자는 반드시 멤버 함수로만 정의되어야 한다. 프렌즈 함수로 정의하는 것은 안 된다.
- &&, ||, and comma operator 는 일반적으로 오버로딩하지 않는다.
  - 오버로딩한다면, short-circuit evaluation 대신에 complete evaluation 을 사용한다.

const 함수들

- 함수 이름 뒤에 const 키워드가 붙으면,
  - 그 함수는 멤버 변수들을 변경할 수 없다.
  - 예: int Matrix::get_dy() const;

friend 함수들과 클래스들

- friend 함수들
  - 연산자 오버로딩 함수들이 멤버 함수가 아닐 때, 객체의 private data 를 접근하기 위해서 객체의 public function, 즉 accessor 를 사용하는 것은 매우 번거롭다.
  - 이 때 오버로딩 함수들을 인자로 사용되는 객체의 소속 클래스에 대한 friend 함수로 정의한다. 그러면, private data 를 직접 접근할 수 있다.
    - friend 함수는 멤버 함수가 아니지만, 클래스 정의 안에 포함되어야 한다.
- friend 클래스들
  - class F is friend of class C
    - class F 의 모든 멤버 함수들은 C 의 friends 이다.
    - 반대는 적용되지 않는다.
  - 문법: friend class F (in class definition of C)

References

- 레퍼런스는 포인터와 다르다.

```
int A;
int &B = A;
```

  - 동일한 메모리 변수를 레퍼런스 A 와 B 가 가리킨다.
  - 따라서 A 를 통해서도 B 를 통해서도 같은 변수를 조작할 수 있다.
- 레퍼런스를 리턴하는 함수는 일반적으로 쓸모가 없다.

```
Matrix& sampleFunction(Matrix &m) { return m; }
```

  - 그러나, << 및 >> 연산자를 오버로딩할 때는 유용하다.
  - 다음 두 표현은 동일하다.

```
cout << "Hello, " << "C++!";
(cout << "Hello, ") << "C++!";
```

Matrix.cpp

```
152 ostream& operator<<(ostream& out, const Matrix& obj){
153   out << "Matrix(" << obj.dy << "," << obj.dx << ")" << endl;
154   for(int y = 0; y < obj.dy; y++){
155     for(int x = 0; x < obj.dx; x++)
156       out << obj.array[y][x] << " ";
157     out << endl;
158   }
159   out << endl;
160   return out;
161 }
```

연산자 오버로딩 개요 2

- 할당 연산자 = 의 오버로딩
  - 멤버 함수로만 오버로딩이 가능하다.
  - 컴파일러에 의해서 자동적으로 오버로딩된다. 이 경우에 member-wise copy (or, shallow copy)가 수행된다.
  - 포인터 변수를 멤버 변수로 가진 클래스들에 대해서는 프로그래머가 직접 오버로딩 함수를 작성해 주어야 한다. 이 경우에는 member-wise copy 가 아니라 deep copy 를 수행한다.

```
163  Matrix& Matrix::operator=(const Matrix& obj)
164  {
165    if (this == &obj) return *this;
166    if ((dx != obj.dx) || (dy != obj.dy))
167      alloc(obj.dy, obj.dx);
168
169    for (int y = 0; y < dy; y++)
170      for (int x = 0; x < dx; x++)
171        array[y][x] = obj.array[y][x];
172    return *this;
173  }
```

연산자 오버로딩 개요 2

- increment 및 decrement 연산자
  - increment 및 decrement 연산자는 두 가지 버전이 존재한다.
    - prefix notation: ++x
    - postfix notation: x++
  - 오버로딩 함수를 작성하면, 그것은 prefix notation 만 오버로딩한 것이다.
  - postfix notation 을 오버로딩하기 위해서는, 별도의 함수를 작성해야 한다.
    - 그 함수는 dummy 인자로 integer 인자를 가져야만 한다
    - 예: operator++(int y) → 컴파일러에게 postfix notation 의 overloading 임을 알려주는 방법이다.
- array 연산자 [ ]
  - 오버로딩 함수는 멤버 함수이어야 한다.
  - 이 연산자는 레퍼런스를 리턴해야 한다.