객체 지향 프로그래밍
(Object Oriented Programming)

11장

강사 – 강대기

차례 (Agenda)

- 연산자 오버로딩
- 프렌드 함수
- << 연산자 오버로딩
- 자동 데이터형 변환과 데이터형 캐스트
- 클래스 변환 함수

연산자 오버로딩

- 기본적으로 함수 오버로딩과 같음
- operator *op*(*argument list*)
  - Time Time::operator+(const Time& t) const; // 640쪽
  - coding+fixing → coding.operator+(fixing) // 643쪽
  - t1+t2+t3 → t1.operator+(t2+t3) → t1.operator+(t2.operator+(t3)) // 644쪽

연산자 오버로딩의 제약(644쪽)

- 피연산자 중 하나가 사용자 정의 데이터형
- 원래 연산자의 문법 규칙을 따라야 한다 – 예를 들어 이항 연산자를 단항 연산자로 쓰지 못하며, 연산자 우선 순위도 못 바꿈
- 연산자 기호를 새로 만들 수 없음 – 예. “**”
- 오버로딩할 수 없는 연산자들이 있음 (645쪽, 646쪽 표 11.1)
- 멤버함수로만 오버로딩 가능한 연산자들(646쪽)

프렌드

- 외부에서 클래스 객체의 private 부분에 접근하는 다른 통로
- 프렌드 함수, 프렌드 클래스(15장 1000쪽), 프렌드 멤버 함수(15장 1006쪽)
- 왜?
  - Time A = B*2.75; → A = B.operator+(2.75);
  - Time A = 2.75*B; → ???
- Time operator*(double m, const time& t);
  - 어떻게 Time t 객체의 private 데이터에 접근한단 말인가?

프렌드 함수 생성

- 클래스 선언 안에 원형을 넣는다
  - friend Time operator*(double m, const time& t);
  - 멤버 함수가 아님, 그러나 동등한 접근 권한을 가짐
- 함수 정의를 외부에 만든다 (friend 키워드 안 씀)
- 이러한 방법은 OOP 의 철학에 맞는가? (653 쪽) – 멤버함수와 프렌드는 단순히 클래스의 인터페이스 방식

연산자를 오버로딩하는 두가지 방법

- 멤버 함수 (Point 클래스의 예)
  - Point Point::operator+(const Point& p) const;
- 프렌드 함수 (Point 클래스에 수를 곱하는 예)
  - Point Point::operator+(double m, const Point& p);

<< 연산자 오버로딩

- ostream 클래스를 바꾸는 것은 좋지 않다
  - Time 클래스에 대한 프렌드 함수
- 첫번째 버전 (655쪽) → 참조 반환 안함, void
- cout << x << y 에 대해서는 동작 안함
  - (cout <<x) << y 이기 때문
- 두번째 버전 (657쪽, 659쪽) → ostream 객체를 참조로 반환

멤버 함수 vs. 프렌드 함수

- T1 = T2 + T3;
  - T1 = T2.operator+(T3);
  - T1 = operator+(T2,T3);
- 둘 다 쓰게 되면, 모호해져서 에러가 남

예: 벡터 클래스 (664쪽)

- 소스 (667쪽부터)
- getter / setter
  - shove.set(100,3000);
  - shove = Vector (100,3000);
- 상태 멤버 – 플랙과 같은 것으로, 그 클래스 객체의 상태를 반영 (극 좌표냐, 직각좌표냐)
- 방어적 프로그래밍 (675쪽)
- 산술 연산자 오버로딩
  - 덧셈: 두 좌표계를 모두 고려해야 한다 (676쪽)
  - 곱셈: double 값과 곱하는 경우, 인라인 프렌드 함수 사용
  - 마이너스 부호: 이항 마이너스와 시그너춰가 다르므로 오버로딩한 걸 다시 한번 오버로딩 (678쪽)
- Random Walk (680쪽)

내장 데이터 형 변환 복습

- 내장 데이터 형 변환
  - long count = 8; // 8L
  - double time = 11; // 11.0
  - int side = 3.33; // 3 (경고)
  - int* p = 10; // 에러
  - int* p = (int*)10;

암시적 데이터형 변환

- 파운드 스톤 변환 프로그램 (115쪽)
- 단일 변수를 가지는 생성자
  - Stonewt(double lbs);
  - Stonewt myCat;
  - myCat = 19.6; // 암시적 데이터 형 변환
- 오히려 헛갈리고 버그를 발생할 수 있음
- explicit Stonewt (double lbs) // 암시적 데이터 형 변환 금지
  - Stonewt myCat;
  - myCat = 19.6; // 암시적 데이터 형 변환→ explicit 에 의해 에러 발생
  - myCat = Stonewt(19.6);
  - myCat = (Stonewt)19.6;
- Stonewt(double) 생성자와 다른 데이터 형 (int)
  - Stonewt Jumbo(7000); // int 를 double로 변환 Stonewt(double) 사용
  - Jumbo = 7300; // int 를 double로 변환 Stonewt(double) 사용

변환 함수

- 역으로 변환은 가능한가?
  - Stonewt wolfe(285.7);
  - double host = wolfe;
- 가능하며, 생성자를 사용하는 게 아니라 변환 함수를 사용
- 강제 변환
  - double host = double (wolfe);
  - double host = (double) wolfe;
- 암시적 변환은 변환 함수를 통해 가능
  - operator typename();
  - 클래스의 메쏘드
  - 리턴 형이 없음
  - 전달 인자가 없음
  - operator int();, operator double();
- 문맥에 따라 자동으로 적용되고, 애매하면 에러 (696쪽)
  - cout << poppins; (에러)
  - double p_wt = poppins; (적용)
  - long gone = poppins; (에러)
  - long gone = (double) poppins; (강제 데이터 형 변환)
  - long gone = int (poppins); (강제 데이터 형 변환)

암시적 변환 자동 수행의 문제점

- 암시적 변환이 드러나지 않는 문제점
  - int ar[20];
  - Stonewt temp(14,4);
  - int Temp = 1;
  - cout << ar[temp] << endl;
- 해결책 중 하나
  - Stonewt::operator int() {return int(pounds+5);}
  - → int Stonewt:: Stone_to_Int() {return int(pounds+5);}
- 요약 – 698쪽~699쪽

변환과 프렌드

- 덧셈 추가 – 덧셈 연산자 오버로딩
  - 멤버 함수 또는 프렌드 함수 중 택일 (699쪽)
    - Stonewt 객체 = Stonewt 객체 + Stonewt 객체
    - 멤버 함수 : jennySt.operator+(bennySt)
    - 프렌드 함수 : operator+(jennySt, bennySt)
  - Stonewt(double) 생성자
    - Stonewt 객체 = Stonewt 객체 + double 형
    - jennySt.operator+(kennyD) // 멤버 함수 전달 인자인
    - operator+(jennySt, kennyD) // kennyD가 변환됨
    - operator double 멤버 함수가 있다면 애매함 발생
  - 다음은 프렌드 함수만 가능
    - Stonewt 객체 = double 형 + Stonewt 객체
    - pennyD가 먼저 변환되어 멤버 함수를 호출하지는 않는다
    - 따라서 operator+(pennyD, jennySt)

double 과 객체의 덧셈 구현을 위한 선택 (701쪽)

- 프렌드 함수와 생성자
  - operator+(const Stonewt&, const Stonewt&)
  - Stonewt(double) 생성자
  - 프로그램이 짧아지나 시간/메모리 부담
- 멤버 함수와 프렌드 함수
  - Stonewt operator+(double);
  - operator+(double, const Stonewt&)
  - 프로그램이 길지만, 속도가 빠름