Chap02

/*
#개념 정리
1.연산자 오버로딩
:기존에 정의된 연산자를 새로운 의미로 정의하는 것을 말한다.
이는 사용자가 정의한 새로운 데이터타입에 대해서 연산자 동작을
사용자가 직접 정의할 수 있게 한다.
연산자 오버로딩은 멤버함수처럼 동작하므로 프로토타입은 클래스 내부에서,
정의는 외부에서 진행한다.

연산자 오버로딩 정의는 클래스 public에서 정의되며
프로토타입은 리턴형 operator[연산자](const 클래스& other);와 같다.

2.throw 시그널을 catch하는 문법
먼저, <stdexcept> include한다.
그 다음, throw가 발생하는 코드에서
throw invalid_argument(string 문자열); 예외 발생 원인을 문자열로 작성한다.

try-catch구문에서 throw 시그널 발생시 catch가 받아서 예외를 처리한다.
예외 문구는 멤버함수인 what()을 사용하면 문자열이 출력된다.

3.Explicit declaration VS Implicit declaration
:Explicit은 내가 초기화하는거고, Implict는 컴파일러가 대신 자동으로 해준다.

4.구조체에서는 생성자, 소멸자가 없었는데 C++에서는 클래스에 의해 생성자와 소멸자가 존재한다.
생성자의 종류는 여러 개이지만, 소멸자는 오직 한 개인게 특징이다.
여러 개의 생성자 중에서 Conversion constructor를 살펴보자.

5.(중요)Conversion constructor(변환 생성자)
변환 생성자는 단 한개의 매개변수만 갖는 생성자이다.
입력된 type을 class type으로 암시적으로 변환할 수 있게 하는 특수한 생성자이다.
변환 생성자에서 매개변수의 type은 입력된 type에서 변환될 type에 해당한다.

변환 생성자 호출은
class-type 객체명 = 입력 type;이다.
입력 type을 해당 class-type으로 변환 생성자에 의해 암시적 형 변환이 일어난다.

5-1.explicit가 존재하면
class-type 객체명 = 입력 type;과 같은 코드는 에러가 발생된다.

(참고로, 변환 생성자 사용은 사용이 제한적이므로 신중하게 사용해야만 한다.)

5-2.변환 생성자는 매개변수 하나이다.

6.composition:매개변수에 기본 type이 아닌 사용자 정의 type이 들어가면 composition 되었다라고 부른다.
만약에 클래스가 들어간다면, '클래스가 재료가 되어서 composition 되었다'라고 부른다.

#다시볼 cpp 파일
(꼭 보기)4.cpp, 5.cpp : 변환 생성자 좋은 예시가 있다.

#잘 몰랐던 개념 정리
1.setw()는 한줄에 영향 주는게 아니라,
당장의 오른쪽 출력값 하나에만 영향을 준다. (?)
setw()와 left 같이 사용할 때 주의해서 사용하자.

*/

Chap03
/*
#3강 강의 정리:this, static
 1.this는 호출되는 객체의 포인터이다.
 따라서, this 사용시 Arrow operator를 사용하여 member에 접근해야한다.
 2.*this는 객체 자기 자신이 된다.
 3.(체크)this 포인터는 static, friend에서 사용불가능하다.
 4.Cascaded Function Calls는 리턴값이 다른 함수를 호출하게 하는 방법이다.
 따라서, *this를 리턴하여 나 자신을 호출하여 다른 멤버함수를 연속해서 사용할 수 있게 된다.
 5.static의 소유권은 class에 있다.
 static 멤버변수는 클래스 밖에서 정의된다!(안에서 정의하면 const int가 아닌이상 컴파일에러가 발생한다.)
 6.static 멤버변수는 단 한번만 초기화된다.
 7.함수호출은 Rvalue가 되어서 어떤 값을 저장할 수가 없다.
 8.int&& a = 5;와 같이 Rvalue를 참조할 수 있는 문법이 있다.
 9.위임생성자: 생성자가 호출될때 다른 생성자로 위임하여 초기화하는 방법이 있다.
 주로, 어떤 객체의 값을 포인터로 초기화할 때 매개변수 생성자를 위임하여 초기화할 수 있다.
 10.

*/
Chap04
/*
#Polymorphism
#새로 알게된 개념
1.cout << (t1+=3).getA() << endl;와 같이 역참조 포인터를 리턴하고 멤버함수로 접근해서 cout 으로 바로 출력이 가능.
2. Tmp t{ 4 };
(t+=3)+=5;
t.display(); 이러한 문법이 가능하다는 것을 알아야한다.
3.
explicit Tmp(int a = 0) //conversion 방지.
: a{ a } {}
와 같이 explicit되어있으면 =로 int에서 Tmp로 불가. 오직 {}만 가능
4.Tmp t1{2,4};와 같이 해당 매개변수 리스트와 대응되는 생성자가 없으면
초기화 불가능해서 컴파일 에러가 발생함!
#강의 개념정리
1.예를 들면 makeSound()를 호출할 때 서로 다른 클래스의 객체에서 각각 다르게 반응하여
moow를 출력하거나 meow를 출력할 수 있게 된다.
2.Compile-Time 다형성
:함수오버로딩과 연산자 오버로딩
3.함수 오버로딩은 반드시
(강의) 같은 이름, 같은 매개변수 타입, 같은 리턴타입, 서로 다른 종류의 인자.
4.연산자 오버로딩
이항 연산자(Binary Operator)와 단항 연산자(Unary operator)
이항 연산자는 2개를 피연산자로 하여금 계산하는 연산자이다.
이항 연산자는 스코프에 따라서 전역 오버로딩과 클래스 멤버 오버로딩으로 나누어져있다.
전역 오버로딩이 클래스 private 멤버를 사용시 클래스 내부에 friend 선언이 필요하다.
클래스 멤버 오버로딩은 클래스 자기 자신과 매개변수로 하나만 받게된다.

이항 연산자 전역 오버로딩
friend return-type operator 연산자 (arg1, arg2);선언
return-type operator 연산자 (arg1, arg2){} 정의
이항 연산자 클래스 멤버 오버로딩
class-type operator 연산자 (const class-type& arg1)
5.연산자 오버로딩 리턴타입

```
Tmp operator+(const Tmp& right) { return Tmp{a + right.a}; }
Tmp operator-(const Tmp& right} { return Tmp{a - right.x}; }
bool operator==(const Tmp& right} { return (x == right.x); }
```

```
bool operator!=(const Tmp& right} { return (x != right.x); }
Tmp& operator++(){ ++x; return *this; } //자기자신, prefix
Tmp& operator+=(){ x += right.x; return *this; } //자기자신
Tmp operator++(int) { Tmp temp{*this}; x++; return temp; } // postfix

friend ostream& operator<<(ostream& out, const Tmp& right){
      out << right.x << "-" << right.y << endl;
      return out;
   }
(여기서 Tmp가 const가 아님!)
friend istream& operator>>(istream& in, Tmp& right){
      in >> right.x;
      in.ignore(10,'-');
      in >> right.y;
      return in;
   }
int& operator[](int index){ return p[index]; }
int operator[](int index){ return p[index]; }



5.단항 연산자
:피연산자 1개를 갖는 연산자를 의미한다.
단항 연산자 오버로딩은
#헷갈린 개념
1.Tmp t1();
t1.print(); 가능
Tmp* pt1 = &t1;
t1->print(); 가능
(*t1).print(); 가능

*/
```

Chap05