오버로딩

남동강

오버로딩(overloading)이란?

- 함수/생성자 오버로딩과 연산자 오버로딩으로 구분 할 수 있다

- 함수/생성자 오버로딩
  - **함수/생성자명은 같지만** 매개 변수의 **자료형이나 수가 다른 함수**의 선언을 허용하여 여러 기능을 제공

- 연산자 오버로딩
  - 기본 자료형이 아닌 클래스 타입, **사용자 정의 클래스**에 **연산자를 재정의** 하여 사용하게 한다

생성자 오버로딩

```
class MyClass
{
public:
  MyClass();
  MyClass(int);
  MyClass(int, int);
  MyClass(char);
};
```

- **주의**
  - 기본 생성자는 선언하지 않아도 기본으로 지원하지만 **생성자 오버로딩을 할 경우 기본 생성자는 직접 선언해 주어야 사용이 사용이 가능**하다

함수 오버로딩

```
void swap(int& l, int& r)
{
  int temp = l;
  l = r;
  r = temp;
}

void swap(double& l, double& r)
{
  double temp = l;
  l = r;
  r = temp;
}
```

함수 오버로딩

- 함수 오버로딩**할 수 없는 경우**
  - 반환 값만 다르다

```
int result()
{
  return 0;
}

char result()
{
  return 'a';
}
```

함수 오버로딩

- 함수 오버로딩**할 수 없는 경우**
  - 형식이 같은 static(정적) 멤버 함수

```
class MyClass
{
  static void func() {}
  void func() {}
};
```

  - 매개 변수를 const(상수)로 구분

```
void func(int param) { … }
void func(const int param) { … }
```

  - 매개 변수 형식이 배열과 포인터

```
void func(int* param) { ... }
void func(int param[]) { ... }
```

함수 오버로딩

- 함수 오버로드 고려사항

함수 선언 요소	오버로드 사용 여부
함수 반환 형식	X
매개 변수의 수	O
매개 변수의 형식	O
줄임표의 존재 여부	O
typedef 이름 사용	X
지정하지 않은 배열 범위	X

연산자 오버로딩

```
class Point
{
	private:
	int m_nX, m_nY;

public:
	Point(int x, int y) : m_nX(x), m_nY(y) {}

	void Show()
	{
		std::cout << "x:" << m_nX << ", y:" << m_nY << std::endl;
	}
};

int main()
{
	Point p1(1, 2);
	Point p2(3, 4);
	Point p3 = p1 + p2;
}
```

- **오류(활성) 이러한 피연산자와 일치하는 "+" 연산자가 없습니다.**

연산자 오버로딩

```
class Point
{
    …

    Point operator+(Point& point)
    {
        return Point(m_nX + point.m_nX, m_nY + point.m_nY);
    }
};
int main()
{
    …

    p1.Show();
    p2.Show();
    p3.Show();
}
```

- +연산자 오버로딩으로 에러 없이 정상적으로 내용이 출력된다
- **P1 + p2는 컴파일 시에 p1.operator+(p2)로 해석**이 되기 때문에 p1.operator+(p2)로 작성 하여도 정상적으로 작동한다