참조 : 공간의 공유 , 새로운 메모리 할당 x

★ 할당된 공간이 있어야 선언 가능

ex) int& refn ; (x)

참조변수 사용 시 & 사용 x

ex) refn = 5 ;

참조 변수 배열 불가능

이중 참조 가능

ex) int &r = refn ;

★ 포인터 참조 변수

double *p = &a

double * &pp = p

참조변수 지역변수 특징

이름만 삭제, 메모리 삭제 x

매개변수 참조로 객체 전달

→ 객체가 새로 생성 x, 생성자/소멸자 실행 x

★ 실 매개변수 값 변경 가능 , 값에 의한 전달과 차이점

참조 리턴

```
Char& find(char s[], int index)
{ return s[index] ; }
```

⇒ find(name, 1) → name[1]

복사 생성자

① 얕은 복사 (자동)

단순 대입

② 깊은 복사 (수동)

동적할당 시 새롭게 동적할당

나머지는 대입

```
class Circle
{
    Circle (const Circle& c) ;
}
```

필수 참조

```
Circle b(a)
= Circle b = a ;
```

함수 중복 - 오버로딩

반환자료형 상관 x, 매개변수의 개수, 자료형으로 구분

디폴트 매개변수

```
Void star (int a=5) ;
```

매개변수를 적지 않으면 a=5

매개변수 주면 a를 그 값으로 초기화

★ 디폴트 매개변수는 뒤에서부터 써야 함

★ 함수 중복의 모호성 ★ 시험

① 자동 형 변환 ★

```
float AA (float a)
double AA (double a)
```

② 참조 매개변수 ★

```
int add (int a, int b)
int add (int a, int &b)
```

③ 디폴트 매개변수 ★

```
Void msg (int id)
Void msg (int id, String s="")
```

Static : 프로그램 시작부터 끝까지 데이터 유지

Static 지역변수 : 데이터는 유지 , 해당 지역에서만 사용 가능

★ Static 멤버

맨 앞에 Static 붙이기 동일한 클래스의 객체끼리 공유

★ 전역 공간에서 선언, 초기화 해야 한다.

ex) int Person :: shareMoney = 10 ;

★ Static 멤버 this 포인터 사용 불가

Static 멤버 접근 방법 3가지

① 클래스명 ::

② 객체.

③ 객체주소 →

Static 멤버함수

객체 생성 이전에 static 멤버에 접근하기 위함

★ non-static 멤버에 접근 불가능 ★ 시험

프렌드 - 일부 캡슐화 완화

클래스 외부 함수를 멤버함수 취급 → 클래스 내 멤버 접근 가능

① 일반 전역함수 프렌드화

```
class Circle {
public:
    friend int Add(Circle& op1, Circle& op2)
};
```

② 다른 클래스의 멤버함수 프렌드화 가능

```
class {
public:
    friend int Rect::Add(Circle &op1, Circle& op2)
};
```

③ 클래스 자체 프렌드화 가능

```
class {
public:
    friend Rect;
};
```

★ 프렌드화 시 어디 소속인지 확실하게 선언

★ 시험 ★ 클래스 선언 전에 프렌드한 함수가 멤버에 접근하는 것을 방지하기 위해 클래스 원형 먼저 선언

ex) class Rect;

연산자 중복

↳ 함수 중복을 이용

. .* :: ?: 연산자 중복 불가능

① 클래스 멤버함수로 구현

② 외부 함수로 구현, 프렌드화

- 이항 연산자 ex) + 연산자

  c = a + b (a가 객체일 경우)

  ① c = a.operator + (b)

  클래스 내 멤버함수로 선언

  Color operator + (Color op1);

  ② c = operator + (a,b)

  일반 전역함수 프렌드화

  friend operator + (Color op1, Color op2)

- 단항 연산자

  전위 연산자 ++op

  op.operator ++ ( );

  후위 연산자 op++

  op.operator ++ (int);

  ↳ int x

  참조 리턴시 연산자 연속으로 사용 가능

Complex 시험 무조건 출제

소스코드 올려주심

상속

```
class Point {
};
class ColorPoint : public Point {
};
```

★ 생성자는 기본 생성자부터 차례대로 실행

소멸자는 생성자의 역순

업캐스팅 ★ 객체 포인터 자료형만 가능

```
ColorPoint cp;
ColorPoint *pDer = &cp;
Point * pBase = pDer;
```

업캐스팅 : 자동 형 변환 / 강제 형 변환) 가능

기본 ← 파생

★ 파생 포인터로 모든 멤버 접근가능

기본 포인터로 기본 멤버만 접근 가능

다운캐스팅

```
pDer = (ColorPoint *)pBase;
```

강제 형 변환 필수

업캐스팅 이후에 다운캐스팅 가능

(↳ 태생부터 파생으로 만들어져야 하기 때문)

파생클래스의 생성자 지정

지정하지 않으면 기본클래스의 기본생성자 호출

지정하면 지정한 기본클래스의 생성자 호출

상속 접근지정자 (생략 시 private로 상속)

private : 기본클래스의 접근지정자 모두 private로 상속

protected : protected, public 을 protected로 상속

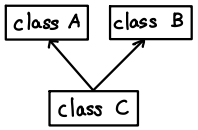
public : 접근지정자 그대로 상속

★ ex) private로 상속 시

파생클래스에서 기본클래스의 private, public 멤버 접근 가능

main 문에서 기본클래스 멤버 모두 접근 불가능

다중 상속




```
class C : public A, public B {
};
```

콤마로 나열, 접근지정자 생략 시 private

★ 문제점 : 멤버가 중복될 수 있다

해결 방법 : 가상 상속 (virtual)

파생클래스 객체 생성 시 기본클래스의 멤버를 오직 1번만 할당, 중복되면 그 공간을 공유함

ex) class C : virtual public A, virtual public B {
};

정적 바인딩	동적 바인딩
컴파일 시 구성요소의 성격이 결정	프로그램 실행 시 성격이 결정
속도 ↑	속도 ↓ 적응성 ↑
	(virtual 키워드)