**객체지향프로그래밍 :4차시**

중요 문단은 \*표시 남김.

**클래스**

객체를 만들기 위해 설계된 틀이나 설계도.  
객체도 아니고 실체도 아니지만, 객체(인스턴스)를 찍어내는 듯한 역할을 수행함.  
(멤버 변수와 멤버 함수를 선언함.)

클래스 작성법 : class ClassName의 형태로 선언, 멤버 변수와 멤버 함수를 클래스 내에 선언함.  
선언 시에 접근지정자(private, pubic, protected)를 지정하여 줌. (Default : private)  
클래스 구현부에 정의된 멤버 함수를 구현함.

* Circle.cpp
* Rectangle.cpp

**객체**

대상으로 나타낼 수 있는 세상의 모든 것이 객체.

객체는 일부 요소가 공개되어 외부와 상호작용함.  
(ex. On/off 버튼, 밝기 조절, 채널 조절, 음량 조절 등의 멤버 함수와 멤버 변수, 파라미터.)

객체는 생성될 때 클래스의 모양을 그대로 가지고 생성됨. 멤버 변수와 함수로 구성되며, 하나의 클래스에서 여러 객체가 생성될 수 있음. 객체는 별도의 공간에 생성됨.

**클래스/인스턴스 붕어빵 비유**

클래스를 붕어빵 틀, 객체(인스턴스)를 붕어빵으로 나타낼 수 있음.  
붕어빵의 경우 붕어빵 틀의 모양대로 찍어져 나오지만, 입력값(앙금 등)에 따라 서로 다른 객체가 될 수 있음.

**캡슐화**

객체를 캡슐로 싸듯이 보호하여, 객체 내 데이터의 보안 및 보호, 외부 접근 제한 등을 목적으로 함

**객체 이름 생성 및 접근 과정**

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

**생성자**

객체 생성 시점에 자동으로 단 한 번 호출되는 멤버 함수  
클래스 이름과 동일한 이름을 가짐.

int main() {

Rectangle rect;

rect.width = 3;

rect.height = 5;

cout << "사각형의 면적은 " << rect.getArea() << endl;

}

생성자는 리턴 타입을 선언하지 않음.

생성자는 한 클래스 내에 여러 개가 존재항 수 있으며, 중복된 생성자 중 하나만 실행할 수 있음.  
생성자가 선언되어 있지 않다면, 컴파일러가 매개 변수를 가지지 않는 기본 생성자를 자동으로 선언함.

생성자는 꼭 있어야 하는가?  
생성자가 하나라도 선언되어 있다면, 컴파일러는 기본 생성자를 선언하지 않으므로 컴파일 에러가 발생함. (다음 쪽 이미지)

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

**위임 생성자**

Circle::Circle()

{

radius = 1;

cout << "반지름 " << radius << " 원 생성" << endl;

}

Circle::Circle(int r)

{

radius = r;

cout << "반지름 " << radius << " 원 생성" << endl;

}

Circle::Circle() : Circle(1) {} // 위임 생성자

Circle::Circle(int r)

{

radius = r;

cout << "반지름 " << radius << " 원 생성" << endl;

}

위임 생성자는 중복 작성된 코드를 간소화해 줌. 두 코드는 같은 기능을 하지만, 위임 생성자를 사용하면 보다 가독성 및 직관성이 뛰어나게 됨.

**지역 객체와 전역 객체의 생성과 소멸 순서 \***

전역 객체 생성 -> 지역 객체(main) 생성 -> 지역 객체 생성

텍스트, 스크린샷, 폰트, 도표이(가) 표시된 사진

자동 생성된 설명

**접근 지정자**

**Private** – 동일한 클래스 내의 멤버 함수만 접근 가능.   
아무 접근지정자도 지정하지 않았을 때의 디폴트 값.  
멤버 변수의 경우 private로 지정하는 것이 바람직함.

**Public** – 모든 다른 클래스에서 접근 가능.

**Protected** – 클래스 자신과 상속받은 자식 클래스에만 허용.

class Sample {

private:

// private 멤버 선언

public:

// public 멤버 선언

protected:

// protected 멤버 선언

};

캡슐화를 시켜줌.

접근지정자는 중복 사용 가능함.

class Sample {

**private:**

int x, y;

public:

Sample();

**private:**

bool checkXY();

};

**함수 호출에 따른 시간 오버헤드**

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

때문에, 작은 크기의 함수를 호출하면, 함수 실행 시간에 비해 호출을 위해 소요되는 부가적인 시간 오버헤드가 상대적으로 크다.

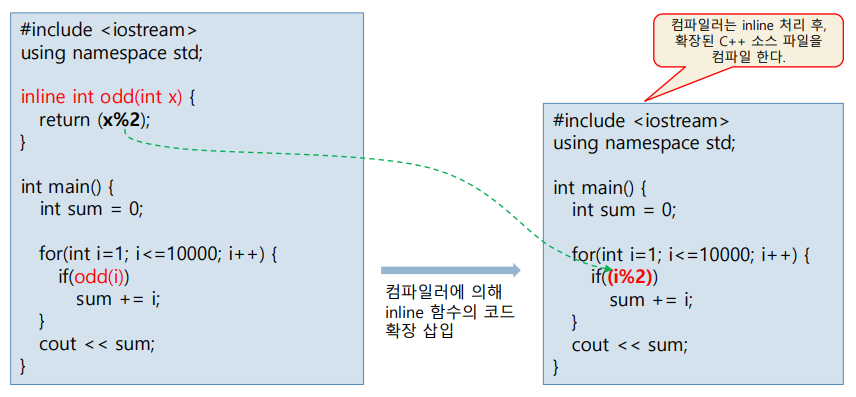
텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명이런 짓 하면 10000번 돌아가서 오버헤드 많이 걸림.

**인라인 함수**

매크로 같이, 인라인 함수가 호출된 위치에 코드를 삽입함.   
이후 인라인 함수는 사라짐.

인라인 함수는 함수 호출에 따른 오버헤드가 존재하지 않고, 프로그램 실행 속도 측면에서 뛰어남.  
단, 컴파일된 전체 코드의 크기가 약 30% 증가함. 때문에, 자주 호출되는 **짧은 코드**에 사용하면 좋음.



또한, 인라인은 컴파일러에게 주는 요구 메시지이고, 컴파일러가 요구를 수용할지 결정하므로 긴 함수나 static 변수, 반복문, switch문, goto문 같은 경우 수용 하지 않음.

**자동 인라인 함수**

클래스 선언부에 구현된 멤버 함수의 경우, 인라인으로 선언하지 않아도 컴파일러에 의해 자동으로 인라인 함수로 처리됨. (생성자도 자동 인라인 함수 처리됨.)

**구조체**

상속, 멤버, 접근 지정이 모두 클래스와 동일하지만, 구조체의 디폴트 접근 지정자는 **public**이다.

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

구조체 객체 생성 시, C언어처럼 struct 키워드를 입력하지 않아도 됨.  
텍스트, 폰트, 화이트이(가) 표시된 사진

자동 생성된 설명

**바람직한 C++프로그램 작성법 \***

1. 클래스 분리 저장
2. 선언부는 헤더 파일로 분리
3. 구현부는 cpp파일로 저장
4. main()은 따로 분리

**#ifndef**

?뭐더라

**#define**

재선언된 include 무시