# C++ 객체지향 프로그래밍 I

(2주차)

# 학습개요

- 학습 목표
  - C++ 클래스 정의하고 객체 생성해 활용할 수 있다.
  - C++ 클래스 상속 메커니즘을 활용할 수 있다.
- 학습 내용
  - 클래스와 객체
  - 상속
  - 실습

# 객체지향 프로그래밍

- 객체지향 프로그래밍 : 객체를 중심으로 한 프로그래밍
- 객체 = 상태(데이터) + 행위(메서드)
- 클래스: C++에서 지원하는 객체를 정의하기 위한 도구로 IS-A 관계에 따라 계층 간 관계를 구성하여 특성을 공유함
- 클래스를 계층적으로 구성하여 기반 클래스(base class or superclass)를 파생 클래스(derived class, subclass)에서 상속받아 사용

# 클래스 정의 및 객체 생성 (1)

- 클래스에 소속된 변수와 함수를 각각 멤버 변수와 멤버 함수라고 함
- 클래스 멤버
  - 멤버 변수: 객체의 속성을 정의함
  - 멤버 함수: 객체의 행위를 정의함
- 인스턴스: 클래스를 통해 정의한 데이터 항목(객체)
- 생성자와 소멸자
  - 생성자: 클래스의 객체를 초기화하는 역할을 수행하는 함수
  - 소멸자: 객체를 위해 할당된 메모리를 해제하는 역할을 수행하는 함수
- 클래스는 생성자 함수를 한 개 이상 포함하며, 소멸자 한 개를 가짐

```
class MyClass{
    int _variable;
public:
    MyClass() { }
    ~MyClass() { }
    void method() { ... }
};
...
MyClass myObject;
```

# 클래스 정의 및 객체 생성(2)

```
class className
{
    ...
    memberList;
    ...
};
```

- className
  - 클래스 이름으로, 이 클래스의 인스턴스를 선언하기 위한 자료형으로 사용함
  - 일반 자료형과 같이 컴파일러가 형 검사(type checking)를 수행함
- memberList
  - 클래스의 멤버(variables, functions, nested classes, enums, bit fields 등)와 프렌드(friend)를 선언함
  - 클래스 안에서는 명시적으로 데이터 초기화를 할 수 없으며, 멤버 변수의 초기화는 생성자 함수에서 수행함
  - 비정적(nonstatic) 멤버로 클래스 자신은 포함할 수 없지만, 포인터 또는 레퍼런스를 통해 클래스 자신을 포함하는 것은 가능함

# 클래스 정의 및 객체 생성 (3)

- 클래스 멤버의 접근 권한
  - 클래스의 멤버(변수 또는 함수)는 키워드 private, protected, public을 사용하여 접근 권한을 지정함
  - private: 클래스의 멤버 함수에서만 접근 가능 (기본 접근 지정자)
  - protected: 클래스의 멤버 함수와 해당 클래스로부터 파생된 클래스의 멤버 함수에서 접근 가능
  - public: 어느 장소에서나 제한 없이 접근 가능
  - 선언된 접근 지정자는 새로운 접근 지정자가 나올 때까지 유효함
  - 일반적으로 클래스를 정의할 때 데이터 멤버는 private 또는 protected로 선언하여 외부로부터 보호함
  - 멤버 함수는 클래스 외부에서 호출할 필요가 없는 함수는 private로 선언하고, 클래스 외부에서 호출이 필요한 함수는 public으로 선언함
  - 생성자와 소멸자는 반드시 public으로 선언함

- 클래스 인스턴스 생성
  - 클래스에 의해 실제 메모리를 할당받은 구체화된 것
  - 인스턴스와 객체를 혼용하여 쓰기도 함
  - 클래스 이름과 변수명을 이용하여 선언함
  - 각 인스턴스마다 멤버 변수에 대한 메모리가 별도로 할당되며, 멤버 함수는 한 번만 할당되고 인스턴스 사이에서 공유함

# 멤버 변수 (1)

- 클래스의 멤버 변수(member variable)는 객체의 속성을 정의
- 접근 지정자(private, protected, public)에 따른 접근 규칙이 있고, 인스턴스의 메모리 할당에 따라 정적(static) 멤버 변수와 비정적(nonstatic) 멤버 변수로 구분됨

- 정적 멤버 변수(static member variable)
  - 클래스의 모든 객체(인스턴스)에 대해서 메모리가 단 하나만 할당됨
  - 정적 멤버 변수의 메모리는 주어진 클래스의 객체 인스턴스에 존재하지 않음
  - 정적 멤버 변수의 초기화는 클래스의 외부에서 수행함
  - 정적 멤버 변수도 클래스 멤버 접근 규칙을 따름
  - private 접근 정적 멤버 변수는 클래스 멤버 함수와 프렌드 함수에서만 접근할 수 있음
  - 정적 멤버 변수는 주어진 클래스의 모든 객체 인스턴스에서 공통적으로 유지해야 할 데이터가 있을 때 사용함

```
class MyClass{
    static int _count;
public:
    MyClass() { }
    ~MyClass() {  }
};
int MyClass::_count = 0; // 정적 멤버 변수 초기화
```

# 멤버 변수 (2)

- 비정적 멤버 변수(nonstatic member variable)
  - static 키워드가 없는 일반 멤버 변수
  - 클래스 객체의 인스턴스마다 개별적으로 메모리가 할당됨
  - 생성자에서 초기화를 수행함

```
class MyClass{
    int _data;
public:
    MyClass() {
       _data = 0; // 생성자에서 비정적 멤버 변수 초기화
    }
    ~MyClass() {  }
};
```

# 멤버 함수 (1)

- 클래스의 멤버 함수(member function)는 객체의 행위를 정의함
- 멤버 함수 역시 비정적 멤버 함수와 정적 멤버 함수로 구분
  - 기본으로 비정적 멤버 함수며, static 키워드를 통해 정적 멤버 함수로 정의함
  - 비정적 멤버 함수는 멤버 연산자(. 또는 ->)로 호출하며, 동일한 클래스의 멤버 함수에서 호출할 경우에는 멤버 선택 연산자를 생략할 수 있음
- 생성자(constructor)
  - 클래스와 이름이 동일한 함수로, 값을 반환하지 않으며 클래스 객체의 인스턴스가 생성될 때 자동으로 호출됨
  - 멤버 변수의 초기화, 메모리 할당 등을 수행함
  - 클래스의 인스턴스는 클래스 외부에서 선언하므로 접근 지정자를 반드시 public으로 선언해야 함
  - 함수 오버로딩을 이용하여 생성자를 하나 이상 정의할 수 있음 (인자 없는 기본 생성자, 복사 생성자 등)
- 소멸자(destructor)
  - 클래스와 이름이 동일한 함수로, 이름 앞에 틸드(~)를 붙여 생성자와 구별함
  - 생성자와 마찬가지로 값을 반환하지 않음
  - 정의된 클래스의 인스턴스의 메모리가 해제될 때 자동으로 호출됨
  - 주로 포인터 멤버 변수에 동적으로 할당된 메모리를 회수하는 일을 수행함
  - 접근 지정자는 반드시 public으로 선언해야 함

# 멤버 함수 (2)

- this 포인터(this pointer)
  - 비정적 멤버 함수가 실행되는 객체 인스턴스를 가리키는 포인터
  - 정적 멤버 함수와 프렌드 함수에는 존재하지 않음
  - 일반적으로 this 포인터는 멤버 함수를 호출한 함수에 해당 멤버 함수가 속한 객체의 포인터를 반환하는 데 사용함
- 정적 멤버 함수(static member function)
  - 정적 멤버 함수는 주로 정적 멤버 변수를 사용하며, 클래스의 객체를 생성하지 않고 클래스 이름과 범위 연산자( :: )를 사용하여 정적 멤버 함수를 호출할 수 있음
  - 정적 멤버 함수는 this 포인터를 사용할 수 없으며, 가상 함수(virtual function)로 선언할 수 없음
- 상수 멤버 함수(constant member function)
  - const 키워드를 사용하여 정의함
  - 상수 멤버 함수 안에서는 멤버 변수의 값을 변경할 수 없음
- 멤버 함수의 구현
  - 멤버 함수는 클래스 정의 부분 안에서 구현할 수도 있지만, 일반적으로는 클래스 정의 안에서 멤버 함수 원형(prototype)만 정의하고, 클래스 정의 외부에서 멤버 함수를 구현함
  - 클래스 정의는 헤더 파일(*.h)에 저장하고, 멤버 함수 구현은 CPP 파일(*.cpp)에 저장함
  - 클래스 정의와 구현을 헤더 파일과 CPP 파일에 분리하여 작성할 때는 CPP 파일에서 #include 문을 사용하여 클래스를 정의하는 헤더 파일을 포함함
  - CPP 파일에서 각 멤버 함수는 클래스 이름과 범위 연산자를 이용하여 멤버 함수가 속한 클래스를 명시함

# 프랜드 함수

- 클래스의 멤버가 아닌 일반 함수 또는 다른 클래스의 멤버 함수에 클래스의 접근 권한과 무관하게 클래스의 모든 접근(public, protected, private)을 허용하려는 경우 프렌드 함수와 프렌드 클래스를 이용함
- 클래스 입장에서 '친구'를 지정하여, 이 친구가 자신의 모든 것을 사용할 수 있게 함
- 프렌드는 상속되지 않음
- 클래스의 멤버가 아니므로 프렌드 함수에서는 this 포인터를 사용할 수 없음

# 프랜드 클래스

- 프렌드 클래스로 선언된 클래스의 모든 멤버 함수는 프렌드 함수가 됨

```
class MyClass {
    int _data;
public:
    MyClass(int a=0){ _data= a; }
    ~MyClass(){ };
    friend class YourClass;  // friend class
    friend int operator+(YourClass ob1, YourClass ob2);
};
```

- 클래스 YourClass의 모든 멤버 함수는 MyClass 클래스의 프렌드 함수가 되므로 private 멤버 데이터인 _data를 사용할 수 있음

# 객체 배열과 포인터

- 객체 배열: 객체의 배열은 메모리가 연속으로 할당되므로, 배열의 크기만큼 생성자가 호출되고, 블록을 빠져나올 때 소멸자도 같은 횟수만큼 호출됨
- 클래스의 포인터는 해당 클래스의 객체를 가리킬 수 있음
  - 클래스 포인터 선언으로 클래스 객체를 위한 메모리가 할당되는 것이 아니기 때문에 생성자가 호출되지는 않음
  - new 연산자로 클래스 객체를 동적으로 할당할 때 생성자가 호출되고, delete 연산자로 메모리를 회수할 때 소멸자가 호출됨

# 클래스 상속 (1)

- 클래스 상속(inheritance)을 통해 클래스를 계층적으로 정의할 수 있음
  - 기반 클래스에 정의된 일반적인 특징은 상속받아 사용하고, 더 구체적인 특징은 파생 클래스에 추가하여 사용함

```
class derivedClassName : [public, protected, private] baseClassList
{
    // member_list
};
```

- baseClassList : 기반 클래스(base class)를 명시함.
  - 기반 클래스 한 개 이상으로부터 파생 가능하며,  각 기반 클래스 이름 앞에는 접근 권한(public, private, protected)을 명시할 수 있음
- public 상속
  - public 상속에서는 기반 클래스의 각 멤버가 파생 클래스에서 동일한 접근 권한을 가진 멤버가 됨. 즉, 기반 클래스의 private 멤버는 파생 클래스의 private 멤버가 되고, 기반 클래스의 public 멤버는 파생 클래스의 public 멤버가 되며, 기반 클래스의 protected 멤버는 파생 클래스의 protected 멤버가 됨
- private 상속
  - private 상속에서는 기반 클래스의 각 멤버가 기반 클래스의 접근 권한과 관계없이 파생 클래스의 private 멤버가 됨. 즉, 기반 클래스의 private, public, protected 멤버 모두 파생 클래스의 private 멤버가 됨
- protected 상속
  - protected 상속에서는 기반 클래스의 public, protected 멤버가 파생 클래스의 protected 멤버가 됨
  - 기반 클래스의 private 멤버는 파생 클래스의 private 멤버가 됨

# 클래스 상속 (2)

- 기반 클래스의 멤버 함수 중복
  - 기반 클래스에 있는 함수를 필요에 따라 파생 클래스에서 동일한 이름으로 함수 오버리이딩(overriding)해서 사용함
- 기반 클래스 초기화
  - 파생 클래스에서 기반 클래스의 생성자를 사용하여 기반 클래스를 초기화할 수 있음
  - 기반 클래스를 초기화하기 위한 파생 클래스의 생성자 구조

```
derivedClassName(): baseClassName(...)
{
  // 파생 클래스의 객체 초기화
}
```

- 다중상속
  - 기반 클래스를 콤마로 구분 나열하여 기반 클래스 한 개 이상으로부터 파생 클래스를 정의할 수 있음
  - 기반 클래스를 나열하는 순서로 생성자가 호출되고 역순으로 소멸자가 호출됨

# 가상함수

- 가상 함수(virtual function): 함수를 지연 바인딩(late binding) 또는 동적 바인딩(dynamic binding)하도록 해서, 실행 시간 다형성(runtime polymorphism)을 지원함

- 바인딩
  - 변수 이름, 자료형, 크기, 값, 인수 호출 방법 등을 포함하여 모든 프로그램 요소의 속성을 결정하는 것을 말함
  - C 언어는 함수를 호출할 때 어느 함수가 호출될지를 결정하는 바인딩을 컴파일 시간에 결정함
  - C에서는 정적 바인딩(static binding)으로 호출될 함수가 결정됨
  - C++ 역시 대부분의 함수는 기본으로 정적 바인딩으로 결정됨
  - C++에서는 함수 선언 앞에 virtual 키워드를 사용하면 바인딩을 지연해서 호출될 함수를 실행 시간에 결정하게 하며, 이를 동적 바인딩이라고 함

- 가상 함수는 대부분 클래스 상속에서 기반 클래스에 있는 멤버 함수를 파생 클래스에서 동일한 이름, 동일한 인수로 오버라이딩(overriding)할 때 사용함

# 학습정리

- 클래스는 멤버 변수와 멤버 함수를 가진다.
- 클래스의 객체를 초기화하는 역할을 수행하는 다수의 생성자 함수와 객체를 위해 할당된 메모리를 해제하는 역할을 수행하는 한 개의 소멸자 함수를 가진다.
- 비정적 멤버 함수가 실행되는 객체 인스턴스를 가리키는 포인터를 this 포인터라고 한다.
- 정적 멤버 함수는 주로 정적 멤버 변수를 사용하며, 클래스의 객체를 생성하지 않고 클래스 이름과 범위 연산자( :: )를 사용하여 정적 멤버 함수를 호출할 수 있다.
- 클래스의 멤버가 아닌 일반 함수 또는 다른 클래스의 멤버 함수에 클래스의 접근 권한과 무관하게 클래스의 모든 접근(public, protected, private)을 허용하려는 경우 프렌드 함수와 프렌드 클래스를 이용할 수 있다.
- public 상속은 기반 클래스의 각 멤버가 파생 클래스에서 동일한 접근 권한을 가진 멤버가 될 수 있도록 한다.
- 가상 함수는 함수를 지연 바인딩 또는 동적 바인딩하도록 해서, 실행 시간 다형성을 지원한다.