과제3

상속은 기존 클래스의 특성(멤버 데이터, 멤버 함수 등)을 가져와서 새로운 클래스로 만드는 것이다. 새로운 클래스는 다른 특성도 포함해서 만들 수 있다.

기존 클래스 -> 부모 클래스

파생 클래스 -> 자식 클래스

```
#include<iostream>
using namespace std;

class A {
private:
    int a{1};

protected:
    int b{2};

public:
    int c{3};
};

class B :public A {
    void showB() {
        //성공 => protected는 자식에서 사용 가능
        cout << b << endl;
    }
};
```

예시 사진처럼 상속클래스 사용 방법은

자식 클래스 : 접근제한자 부모 클래스

{

자식 클래스 내용

}

이다. 중요한 점은 접근 제한자이다. 상속 관련 접근 제한자에 관해 알기 전에 접근 제한자에 알아보자면

Private : 오직 클래스 안에서만 접근 가능하다. 예를 들어서, 상속을 통해서 부모 클래스의 private 변수를 가져오고 싶다면 부모 클래스에서 get함수를 통해 가져와야한다.

Protected : 자신을 상속하는 클래스까지만 접근 가능.
외부에서는 접근이 안되지만, 상속되어지는 자식 클래스에서는 이를 사용 가능하다. 따라서, 부모 클래스에서 private로 선언되어 있는 멤버 변수를 protected 로 바꾼다면 get함수를 사용하지 않더라도 바로 사용 가능하다.

Public : 어떤 클래스에서든 접근 가능하다.

이제 상속할 때 접근 제한자에 따른 사용법위를 확인해보면

1. 만약 private 로 상속을 받는다면, 아래와 같이 축소된다.

- private -> private
- protected -> private
- public -> private

2. 만약 protected 로 상속을 받는다면, 아래와 같이 축소된다.

- private -> private
- protected -> protected
- public -> protected

3. 만약 public 로 상속을 받는다면, 아래와 같이 축소된다.

- private -> private
- protected -> protected
- public -> public

위 사진을 예시로 이어서 설명하자면 메인함수에서 자식클래스 B에서 객체를 생성하고 객체.a, 객체.b 관련한 것들은 오류가 나고, 객체.c는 오류가 안 날것이다.(c만 public 변수이기 때문)

그리고 부모 클래스에서의 변수를 자식 클래스에서 바로 받을 수 없다.

방법 1 : 변수 get 함수 생성

1. 변수 get 함수 생성

```
#include<iostream>
using namespace std;

class A {
private:
    int a{ 1 };
    int b{ 2 };
    int c{ 3 };

public:
    //get함수를 통해 private변수 접근
    int getA() { return a; }
    void showAdd() {
        cout << a << endl << b << endl << c << endl;
    }
};

class B :public A {

public:
    void showA() {
        cout << getA() << endl;
    }
};
```

방법 2 : private -> protected로 변경

```
#include<iostream>
using namespace std;

class A {
//protected로 선언하면서 자식에서 사용 가능하게 한다.
protected:
    int a{ 1 };
    int b{ 2 };
    int c{ 3 };

public:
    void showAdd() {
        cout << a << endl << b << endl << c << endl;
    }
};

class B :public A {

public:
    void showA() {
        cout << a << endl;
    }
};
```

Friend 클래스 : friend로 선언된 다른 클래스의 private 및 protected 멤버에 접근할 수 있다.

```
#include <iostream>
#include <string>
using namespace std;

class Friend1 {
private :
    string name;

    friend class Friend2;
};

class Friend2{
public :
    void set_name(Friend1& f, string s) {
        f.name = s;
    }
    void show_name(Friend1& f) {
        cout << f.name << "\n";
    }
};

int main(void) {
    Friend1 f1;
    Friend2 f2;

    f2.set_name(f1, "열코");
    f2.show_name(f1);

    return 0;
}
```


클래스 friend1에서 friend class Friend2; 해줌으로써 Friend2의 함수들이 메인함수에서 Friend1의 private 변수에 접근해서 출력이 가능하다.

Friend 함수 : friend 클래스와 마찬가지로 private 및 protected 멤버에 접근할 수 있는 권한을 부여할 수 있다. friend 기능을 클래스단위가 아닌 멤버 함수 단위로 지정해 주는 것이다.

```
#include <iostream>
#include <string>
using namespace std;

class Friend1 {
private :
    string name;

    friend void set_name(Friend1&, string);
};


void set_name(Friend1& f, string s) {
    f.name = s;
    cout << f.name << "\n";
}

int main(void) {
    Friend1 f1;

    set_name(f1, "열코");

    return 0;
}
```


Friend 함수가 메인함수에 쓰여도 private 변수가 잘 출력된다.

Static

Static 키워드는 다양한 곳에 쓰일 수 있다.

1.static 전역변수

전역 변수 앞에 static을 붙이면 해당 변수가 선언된 파일 내에서만 접근 가능하다. 이렇게 하면 다른 파일에서 동일한 이름의 변수가 있을 때 충돌을 방지할 수 있다.

2.함수 내의 static 변수

함수 내에 선언된 static 변수는 호출 될때마다 초기화 되지 않고, 프로그램이 종료될 때까지 그 값이 유지 가능하다. 이렇게 하면 함수 호출 사이에 값을 유지하고 싶을 때 사용하면 유용하다.

3. 클래스이 static 멤버 변수

클래스에서 static으로 선언된 멤버 변수는 클래스 전체에서 공유되며, 클래스의 모든 객체가 이 변수를 공유합니다. 즉, 특정 객체가 아니라 클래스의 속성으로 존재합니다. 이는 모든 객체가 같은 변수를 참조하거나 값을 변경해야 할 때 유용하다.

4.클래스의 static 멤버함수

static 멤버 함수는 클래스의 객체와 독립적으로 동작합니다. 이 함수는 객체가 아닌 클래스 자체에 속하므로, 객체의 멤버 변수에 접근할 수 없고, 오직 static 변수와 다른 static 함수만 접근 가능하다.

Const

Const 키워드는 '상수'를 의미하며 값이 한번 초기화된 후 변경되지 않음을 나타낸다. Const는 코드에서 불변성을 보장하고 예기치않은 값변경을 방지하여 코드의 안정성을 높인다. 변수, 함수, 멤버함수, 참조, 포인터 등 다양한 곳에 쓰인다.

단일 상속

단일 상속은 한 클래스가 단일 부모 클래스로부터 상속받는 형태입니다. 자식 클래스는 부모 클래스의 멤버 변수와 메소드를 상속받을 수 있다. 이는 상속의 가장 기본적인 형태이며, 계층적인 구조를 쉽게 유지할 수 있습니다. 단일 상속의 장점은 하나의 부모 클래스만 상속 받으므로 클래스 간 관계가 단순하고 명확하다.

다중 상속

다중 상속은 자식 클래스가 둘 이상의 부모 클래스로부터 상속받는 형태이다. 여러 부모 클래스의 기능을 동시에 상속받아 사용할 수 있지만, 상속 관계가 복잡해질 수 있으며, 특히 두 부모 클래스에 동일한 이름의 멤버가 있을 경우 모호성 문제가 발생할 수 있다. 그러나 이런 모호성의 문제는 부모 클래스이 이름을 사용해 해결 가능하다.

가상 상속

가상 상속은 다중 상속에서 발생할 수 있는 모호성 문제를 해결하기 위한 방법이다. 일반적인 다중 상속에서 부모 클래스가 동일한 상위 클래스를 상속받을 경우, 자식 클래스가 중복된 상속을 받을 수 있는데, 이를 가상 상속으로 방지할 수 있다.

예시

조부모 클래스

부모 클래스1 : 조부모 클래스

부모 클래스2: 조부모 클래스

자식 클래스 : 부모 클래스1, 부모 클래스2

Virtual 키워드를 사용해 가상 상속을 하는데, 이를 통해 부모 클래스 중 한번만 상이 클래스의 멤버를 상속 받는다.

가상함수

가상 함수는 기본 클래스에서 선언된 함수로, 파생 클래스에서 오버라이딩이 가능하다. 가상 함수를 사용하면, 포인터나 참조를 통해 호출할 때 객체의 실제 타입에 따라 적절한 함수가 호출되는 동적 바인딩(Dynamic Binding)을 구현할 수 있다.

순수 가상 함수

순수 가상 함수는 기본 클래스에서 구현을 제공하지 않고, 파생 클래스에서 반드시 재정의해야 하는 가상 함수이다. 순수 가상 함수는 = 0으로 선언하며, 이를 포함하는 클래스는 추상 클래스가 된다.

추상 클래스

추상 클래스는 하나 이상의 순수 가상 함수를 포함하는 클래스입니다. 추상 클래스는 인스턴스화할 수 없으며, 주로 공통 인터페이스를 정의하고 파생 클래스에서 이를 구현하도록 강제하는 역할을 합니다.