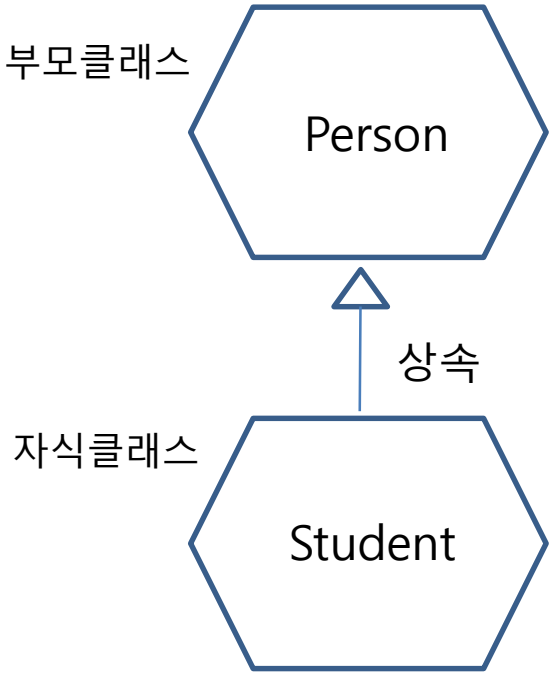
C++_상속, 다형성

Visual Studio 2022

상속(Inheritance)

- 상속이란?

  이미 구현된 클래스를 재사용해서 속성이나 기능을 확장하는 객체 지향 언어의 특성을 말한다.




```
class 클래스이름 : 부모클래스 이름{
    멤버 변수
}
```

```
class Person{
    멤버 변수
};
class Student : public Person{
    멤버 변수
};
```

콜론(:) 1개 사용
public 사용

생성자 상속 및 protected

- 생성자 상속

```
자식 클래스(부모 멤버 변수, 자식 멤버 변수) :
        부모 클래스(부모 멤버), 자식 클래스(자식 멤버) {
}
```

- 접근 지정자

접근 지정자	설 명
public	외부 클래스 어디에서나 접근 할수 있다.
protected	**클래스 내부와 상속관계의 모든 자식 클레스에서 접근 가능**
private	같은 클래스 내부 가능, 그 외 접근 불가

상속의 선언과 활용

- 부모 클래스 정의

```
#ifndef PERSON_H
#define PERSON_H
#include <string>
using namespace std;

class Person {
protected:  //상속받는 클래스만 접근 허용
    string name;  //이름

public:
    Person(string name);  //생성자

    void greet();  //인사하다
    void displayInfo(); //사람의 정보 출력
};
#endif
```

상속의 선언과 활용

- 부모 클래스 구현

```
#include <iostream>
#include "Person.h"

Person::Person(string name) {
    this->name = name;
}

void Person::greet() {
    cout << "안녕하세요. 성명: " << name << endl;
}

void Person::displayInfo() {
    cout << "Person name: " << name << endl;
}
```

상속의 선언과 활용

- 자식 클래스 정의 : 상속

```
#ifndef STUDENT_H
#define STUDENT_H
#include "Person.h"

//Person 클래스를 상속한 Student
class Student : public Person{
private:
    int studentId; //학번

public:
    //생성자 - 부모클래스 멤버 변수 명시함
    Student(string name, int studentId);

    void greet();
    void displayInfo();
};
#endif
```

상속의 선언과 활용

- 자식 클래스 구현 : 상속

```
#include <iostream>
#include "Student.h"

//부모 클래스의 생성자 상속
Student::Student(string name, int studentId) :
    Person(name), studentId(studentId) {}

void Student::greet() { //함수 재정의(override)
    cout << "안녕하세요. 성명: " << name
         << ", 학번: " << studentId << endl;
}

void Student::displayInfo() {
    cout << "Student name: " << name << endl;
}
```

상속의 선언과 활용

- 상속 테스트

```
#include "Student.h"

int main()
{
    //Person 인스턴스 생성
    Person p1("이종범");
    p1.greet();
    p1.displayInfo();

    //Student 인스턴스 생성
    Student st1("이정후", 51);
    st1.greet();
    st1.displayInfo();

    return 0;
}
```

```
안녕하세요. 성명: 이종범
Person name: 이종범
안녕하세요. 성명: 이정후, 학번: 51
Student name: 이정후
```

멤버 함수 재정의(Override)

- 메서드 오버라이드(Override)

  부모 클래스의 멤버 함수를 자식 클래스에서 다시 정의하는 것으로 함수 재정의(Override)라 한다.

Person 클래스

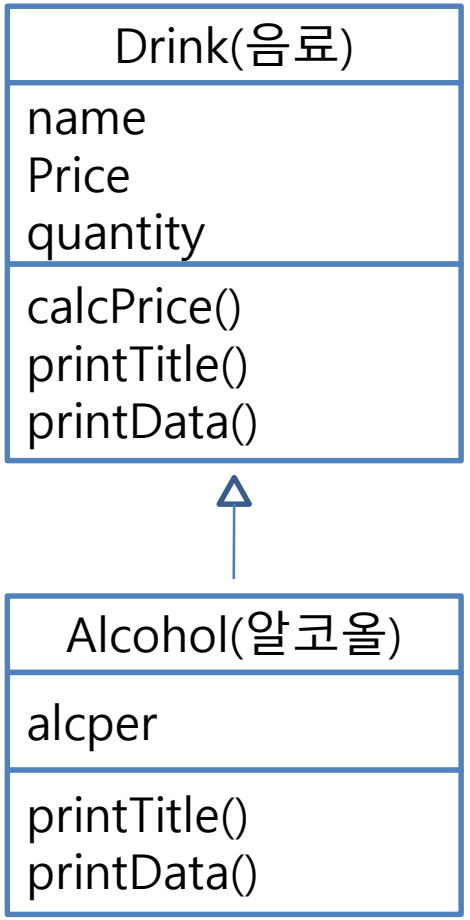
```
void greet() {
    cout << "안녕하세요. 성명: " << name << endl;
}
```

Student 클래스

```
void greet() {
    cout << "안녕하세요. 성명: " << name <<
            ", 학번: " << studentId << endl;
}
```

매출 전표(statement)

- 매출 전표 작성하기



매출 전표(statement)

- 매출 전표 출력 결과

```
============ 매 출 전 표 ============
상품명   가격     수량     금액
커피     2500     4        10000
녹차     3500     3        10500
소주(14.6)       3000     2        6000
***** 합계 금액: 16000원 *****
```

매출 전표(statement)

- 음료(Drink) 클래스 정의

```
#ifndef DRINK_H
#define DRINK_H
#include <string>
using namespace std;

class Drink {
protected:
    string name;    //상품명
    int price;      //가격
    int quantity;   //수량

public:
    Drink(string name, int price, int quantity);

    int calcPrice();  //금액 계산
    static void printTitle(); //제목 출력
    void printData();         //데이터 출력
};
#endif
```

매출 전표(statement)

- 음료(Drink) 클래스 구현

```
#include <iostream>
#include "Drink.h"

//생성자 초기화 목록
Drink::Drink(string name, int price, int quantity) :
    name(name), price(price), quantity(quantity){ }

//금액 = 가격 x 수량
int Drink::calcPrice() {
    return price * quantity;
}

//주의: static을 붙이지 않음
void Drink::printTitle() {
    cout << "상품명\t가격\t수량\t금액\n";
}

void Drink::printData() {
    cout << name << "\t" << price << "\t"
        << quantity << "\t" << calcPrice() << endl;
}
```

매출 전표(statement)

- 알코올(Alcohol) 클래스 정의

```
#ifndef ALCOHOL_H
#define ALCOHOL_H
#include "Drink.h"

class Alcohol : public Drink {
private:
    float alcper; //알콜 도수

public:
    Alcohol(string name, int price, int quantity, float alcper);

    static void printTitle();
    void printData();

};
#endif
```

매출 전표(statement)

- 알코올(Alcohol) 클래스 구현

```
#include <iostream>
#include "Alcohol.h"

//멤버 변수 상속
Alcohol::Alcohol(string name, int price, int quantity, float alcper) :
    Drink(name, price, quantity), alcper(alcper){ }

void Alcohol::printTitle() {
    cout << "상품명(도수[%])\t가격\t수량\t금액\n";
}

void Alcohol::printData() {
    cout << name << "(" << alcper << ")\t" << price << "\t"
        << quantity << "\t" << calcPrice() << endl;
}
```

매출 전표(statement)

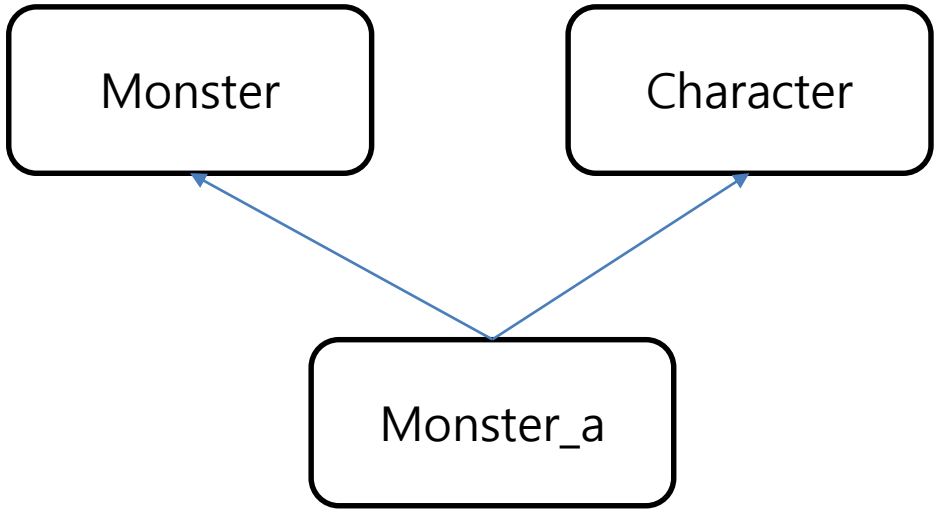
- statementMain.cpp

```
//Drink 인스턴스 생성
Drink coffee("커피", 2500, 4);
Drink tea("녹차", 3500, 3);

//Alcohol 인스턴스 생성
Alcohol soju("소주", 3000, 2, 14.6f);

cout << "============ 매 출 전 표 ============\n";
Drink::printTitle();
coffee.printData();
tea.printData();
soju.printData();

//매출 금액의 합계 계산
int total = coffee.calcPrice() + soju.calcPrice();
cout << "***** 합계 금액: " << total << "원 *****\n";
```

다중 상속(Multiple Inheritance)

- 다중상속(multiple inheritance)

  하나의 파생 클래스가 여러 클래스를 동시에 상속받는 것이다.



다중 상속(Multiple Inheritance)

- Character, Monster 클래스

```
class Character {
public:
    Character() {
        cout << "Character 클래스 생성자" << endl;
    }
    ~Character() {
        cout << "Character 클래스 소멸자" << endl;
    }
};

class Monster {
public:
    Monster() {
        cout << "Monster 클래스 생성자" << endl;
    }
    ~Monster() {
        cout << "Monster 클래스 소멸자" << endl;
    }
};
```

다중 상속(Multiple Inheritance)

- Character, Monster 클래스를 상속받은 MonsterA 클래스

```
class MonsterA : public Monster, Character {
private:
    int location[2]; //좌표 저장

public:
    //기본생성자 : 초기화 목록
    MonsterA() : MonsterA(0, 0) {
        cout << "MonsterA 클래스 생성자" << endl;
        //MonsterA(0, 0); //초기화 되지 않음
    }

    MonsterA(int x, int y) : location{ x, y } {
        cout << "MonsterA 클래스 생성자(매개변수 추가)" << endl;
    }

    void showLocation() {
        cout << "위치(" << location[0] << ", " << location[1] << ")" << endl;
    }
};
```

다중 상속(Multiple Inheritance)

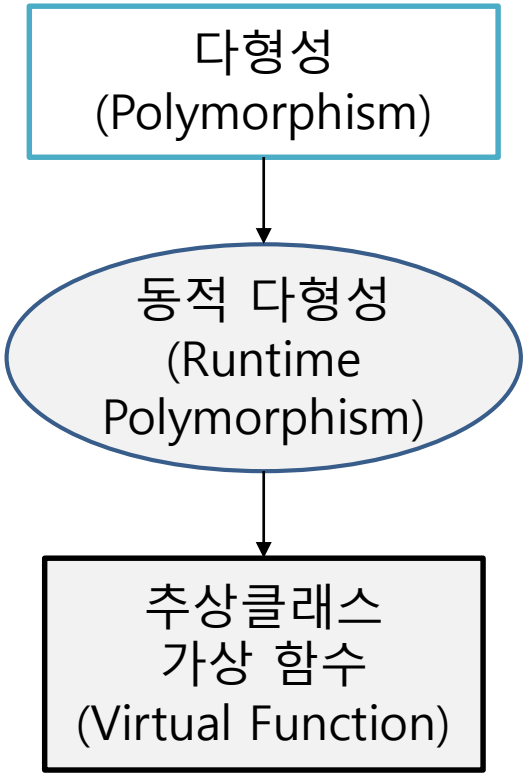
- Character, Monster 클래스를 상속받은 MonsterA 클래스

```
int main()
{
    MonsterA forestMonster;   //기본 생성자 호출
    forestMonster.showLocation();

    MonsterA woodMonster(10, 20); //매개변수가 있는 생성자 호출
    woodMonster.showLocation();

    return 0;
}
```

```
Monster 클래스 생성자
Character 클래스 생성자
MonsterA 클래스 생성자(매개변수 추가)
MonsterA 클래스 생성자
위치(0, 0)
Monster 클래스 생성자
Character 클래스 생성자
MonsterA 클래스 생성자(매개변수 추가)
위치(10, 20)
Character 클래스 소멸자
Monster 클래스 소멸자
Character 클래스 소멸자
Monster 클래스 소멸자
```

다형성(Polymorphism)

다형성이란? 다양한 종류의 객체에게 동일한 메시지를 보내더라도 각 객체들이 서로 다르게 동작하는 특성을 말한다.



가상함수와 동적 결합(Dynamic Binding)

- 다형성에 의해 함수 재정의시 요구 조건
  - 부모 클래스의 멤버 함수가 가상함수(추상함수)로 선언되어야 함
  - **virtual** 키워드를 사용한다.
  - 함수 구현부의 내용은 비워둔다.

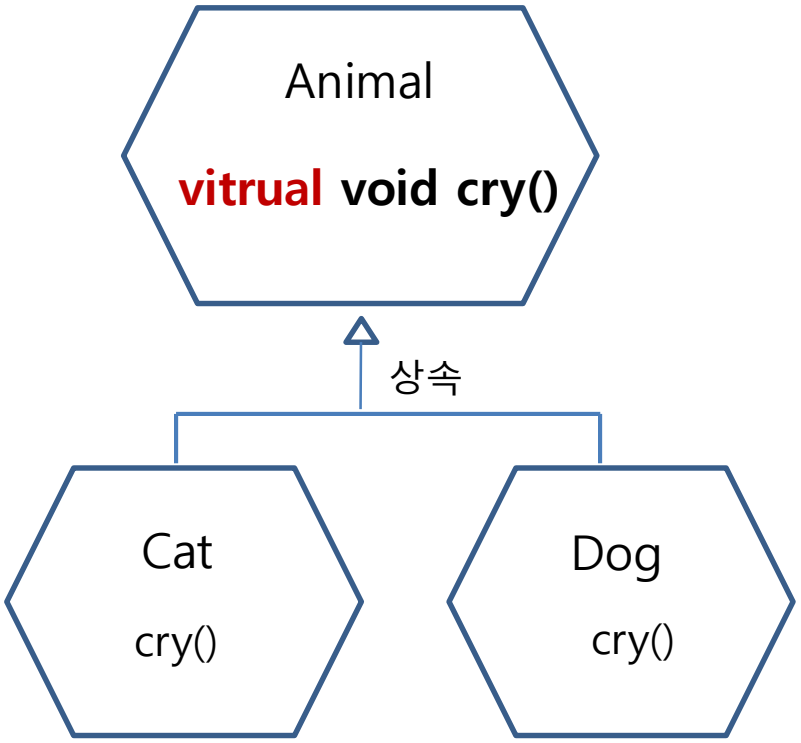
```
virtual cry() = 0
```

- 동적 결합 (Dynamic Binding)
  - 실행시 호출될 함수를 결정하는 것으로 이는 하나의 함수가 여러 클래스에서 오버라이딩 되었을 때 사용한다.
  - 객체 생성시 **new**, 해제 시 **delete** 사용

```
Animal* cat = new Cat     부모클래스 = new 자식클래스(자동 형변환 )
```

가상(Virtual) 함수

- 가상(추상) 함수 사용



가상(Virtual) 함수

- 가상 함수 사용

```
class Animal {
public:
    //Animal() {}  //기본 생성자

    //반드시 virtual로 소멸자 명시함
    //소멸자 없으면 delete시에 메모리 누수 현상 발생함
    virtual ~Animal() {}

    void breathe() {
        cout << "숨을 쉽니다.\n";
    }
    virtual void cry() = 0; //순수 가상 함수
    //virtual void cry() {}
};
```

가상(Virtual) 함수

- 가상 함수 사용

```
class Cat : public Animal {
public:
    void cry() {
        cout << "야~ 옹!\n";
    }
};

class Dog : public Animal {
public:
    void cry() {
        cout << "왈~ 왈~\n";
    }
};
```

가상(Virtual) 함수

- 가상 함수 사용

```
int main()
{
    //정적 객체 생성
    /*Cat cat;
    cat.breathe();
    cat.cry();*/

    //동적 객체 생성
    Animal* cat = new Cat;
    Animal* dog = new Dog;

    cat->breathe();
    cat->cry();

    dog->breathe();
    dog->cry();

    delete cat;  //메모리 해제
    delete dog;

    return 0;
}
```

```
숨을 쉽니다.
야옹~
숨을 쉽니다.
멍~ 멍~
```

다형성 예제

- Person 클래스 정의

```
#ifndef PERSON_H
#define PERSON_H
#include <string>
using namespace std;

class Person {
protected:
    string name;

public:
    Person(string name);

    virtual void greet();       // 가상 함수
    virtual void displayInfo(); // virtual
    virtual ~Person();          // 가상 소멸자(필수)
};
#endif
```

다형성 예제

- Person 클래스 구현

```
#include <iostream>
#include "Person.h"

Person::Person(string name) {
    this->name = name;
}

void Person::greet() {
    cout << "안녕하세요. 성명: " << name << endl;
}

void Person::displayInfo() {
    cout << "Person name: " << name << endl;
}

Person::~Person() {
    cout << "Person 소멸자 호출됨: " << name << endl;
}
```

다형성 예제

- Student 클래스 정의

```
#ifndef STUDENT_H
#define STUDENT_H
#include "Person.h"

class Student : public Person {
private:
    int studentId;

public:
    Student(string name, int studentId);

    void greet() override;          //함수 재정의
    void displayInfo() override;  //오버라이드
    ~Student() override;    //가상 소멸자
};
#endif
```

다형성 예제

- Student 클래스 구현

```
#include <iostream>
#include "Student.h"

Student::Student(string name, int studentId)
    : Person(name), studentId(studentId) {
}

void Student::greet() {
    cout << "안녕하세요. 성명: " << name
         << ", 학번: " << studentId << endl;
}

void Student::displayInfo() {
    cout << "Student name: " << name
         << ", ID: " << studentId << endl;
}

Student::~Student() {
    cout << "Student 소멸자 호출됨: " << name << endl;
}
```

다형성 예제

- StudentMain.cpp

```
void introduce(Person* p) {
    // 다형성: Student 인스턴스면 Student의 greet() 호출
    p->greet();
    p->displayInfo();
}

int main() {
    //동적 인스턴스 생성
    Person* p1 = new Person("이종범");
    Student* s1 = new Student("이정후", 101);

    cout << "[Person 객체]" << endl;
    introduce(p1);

    cout << "\n[Student 객체]" << endl;
    introduce(s1);  // Person 포인터지만 실제는 Student 객체

    delete p1; //메모리 반납
    delete s1;
```

```
[Person 객체]
안녕하세요. 성명: 이종범
Person name: 이종범

[Student 객체]
안녕하세요. 성명: 이정후, 학번: 101
Student name: 이정후, ID: 101
Person 소멸자 호출됨: 이종범
Student 소멸자 호출됨: 이정후
Person 소멸자 호출됨: 이정후
```

다형성 – 매출 전표

- Drink 클래스 정의

```
#ifndef DRINK_H
#define DRINK_H
#include <string>
using namespace std;

class Drink {
protected:
    string name;    //상품명
    int price;      //가격
    int quantity;   //수량

public:
    Drink(string name, int price, int quantity);

    int calcPrice();  //금액 계산
    virtual void printTitle();   //제목 출력
    virtual void printData();   //데이터 출력
    virtual ~Drink();    //가상 소멸자
};
#endif
```

다형성 – 매출 전표

- Drink 클래스 구현

```
Drink::Drink(string name, int price, int quantity) :
    name(name), price(price), quantity(quantity) {
}

//금액 = 가격 x 수량
int Drink::calcPrice() {
    return price * quantity;
}

void Drink::printTitle() {
    cout << "상품명\t가격\t수량\t금액\n";
}

void Drink::printData() {
    cout << name << "\t" << price << "\t"
        << quantity << "\t" << calcPrice() << endl;
}

Drink::~Drink() {
    //cout << "Drink 소멸자 호출됨\n" << endl;
}
```

다형성 – 매출 전표

- NonAlcohol 클래스 정의

```
#ifndef NONALCOHOL_H
#define NONALCOHOL_H
#include "Drink.h"

class NonAlcohol : public Drink {
public:
    NonAlcohol(string name, int price, int quantity);

    void printTitle() override;
    void printData() override;
    ~NonAlcohol() override;
};
#endif
```

다형성 – 매출 전표

- NonAlcohol 클래스 구현

```
#include <iostream>
#include "NonAlcohol.h"

NonAlcohol::NonAlcohol(string name, int price, int quantity) :
    Drink(name, price, quantity){}

void NonAlcohol::printTitle() {
    cout << "상품명\t가격\t수량\t금액\n";
}

void NonAlcohol::printData() {
    cout << name << "\t" << price << "\t"
        << quantity << "\t" << calcPrice() << endl;
}

NonAlcohol::~NonAlcohol() {
    //cout << "NonAlcohol 소멸자 호출됨\n";
}
```

다형성 – 매출 전표

- Alcohol 클래스 정의

```
#ifndef ALCOHOL_H
#define ALCOHOL_H
#include "Drink.h"

class Alcohol : public Drink {
private:
    float alcper; //알콜 도수

public:
    Alcohol(string name, int price, int quantity, float alcper);

    void printTitle() override;
    void printData() override;
    ~Alcohol() override;
};
#endif
```

다형성 – 매출 전표

- Alcohol 클래스 구현

```
//멤버 변수 상속
Alcohol::Alcohol(string name, int price, int quantity, float alcper) :
    Drink(name, price, quantity), alcper(alcper) {
}

void Alcohol::printTitle() {
    cout << "상품명(도수[%])\t가격\t수량\t금액\n";
}

void Alcohol::printData() {
    cout << name << "(" << alcper << ")\t" << price << "\t"
        << quantity << "\t" << calcPrice() << endl;
}

Alcohol::~Alcohol() {
    //cout << "Alcohol 소멸자 호출됨\n";
}
```

다형성 – 매출 전표

- 매출 전표 테스트

```
int main()
{
    //다형성 포인터 배열
    const int SIZE = 3;
    int total = 0;
    Drink* drinks[SIZE];

    //인스턴스 생성
    drinks[0] = new NonAlcohol("커피", 2500, 4);
    drinks[1] = new NonAlcohol("녹차", 3500, 3);
    drinks[2] = new Alcohol("소주", 3000, 2, 14.4f);

    //데이터 출력
    cout << "============ 매출 전표 ==========\n";
    /*for (int i = 0; i < SIZE; i++) {
        drinks[i]->printTitle();
        drinks[i]->printData();
        cout << endl;
    }*/
```

다형성 – 매출 전표

- 매출 전표 테스트

```
//범위 기반 for - auto 사용
for (auto drink : drinks) {
    drink->printTitle();
    drink->printData();
}

//합계 금액
for (int i = 0; i < SIZE; i++) {
    total += drinks[i]->calcPrice();
}

cout << "***** 총 합계 금액 : " << total << "원 *****\n\n";

//메모리 해제
for (int i = 0; i < SIZE; i++) {
    delete drinks[i];
}
```

auto 자료형 키워드

auto 자료형 키워드

- auto 키워드는 변수 선언시에 변수의 타입을 결정하도록 지시한다.
- auto는 변수의 타입을 자동 추론할 수 있다.

```
/*int square(int x) {
    return x * x;
}*/
//inline 함수 - 함수 호출이 일어나지 않음
//프로그램의 실행 속도 저하를 막기 위한 기능
int square(int x) { return x * x; }

int main()
{
    auto ch = 'K'; //문자형
    auto num = 12; //정수형
    auto unit = 2.54; //실수형
    auto* ip = &num;  //정수형 포인터

    cout << ch << ", " << num << ", " << unit << endl;
    cout << *ip << endl;
```

inline 함수는 호출 시 함수 호출 오버헤드 없이, 해당 위치에 함수 본문을 복사해서 삽입한다.

auto 자료형 키워드

- **auto 자료형 키워드**

```
//함수의 리턴 타입
auto value = square(9);
cout << value << endl;

//벡터 자료구조
vector<int> vec = { 1, 2, 3, 4 };

/*for (int i = 0; i < vec.size(); i++) {
    cout << vec[i] << " ";
}*/

//범위 기반 for - int형 대신 auto 사용
//참조로 순회하려면 auto&를 사용함
for (auto& v : vec) {
    cout << v << " ";
}
```

```
K, 12, 2.54
12
81
1 2 3 4
```

연산자 오버로딩(중복)

- 연산자 오버로딩
  - ✓ 연산자를 재정의하여 사용자 정의 클래스로 사용하는 것을 말한다.

함수반환형 **Operator** 연산자 (연산대상){ … }

연산자 오버로딩(중복)

- 객체 연산자 만들기

```
class Point {
    int x, y;

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

    void print() {
        cout << "x = " << x << ", y = " << y << endl;
    }

    //Point&(참조자)를 사용하여 원본은 유지하고 복사본을 리턴함
    Point operator+(Point& p) { //더하기 연산자
        return Point(x + p.x, y + p.y);
    }

    Point operator-(Point& p) { //빼기 연산자
        return Point(x - p.x, y - p.y);
    }
};
```

연산자 오버로딩(중복)

- 객체 연산자 만들기

```
//점 객체
Point p1(1, 2);
Point p2(3, 4);

p1.print(); //출력 함수 호출
p2.print();

//객체 더하기
Point p3 = p1 + p2;
p3.print();

//객체 빼기
Point p4 = p2 - p1;
p4.print();
```

```
x = 1, y = 2
x = 3, y = 4
x = 4, y = 6
x = 2, y = 2
```

연산자 오버로딩(중복)

- 객체의 크기 비교(비교 연산)

```
class Circle {
    double radius; //반지름(실수)
    double const PI = 3.1415; //원주율 상수

public:
    Circle(double radius) {
        this->radius = radius;
    }

    double getRadius() { return radius; }
    //원의 면적 계산
    double getArea() { return PI * radius * radius; }

    bool operator >= (Circle c) { //비교 연산자 함수 생성
        if (this->radius >= c.radius)
            return true;
        else
            return false;
    }
};
```

연산자 오버로딩(중복)

- 객체의 크기 비교(비교 연산)

```
int main()
{
    Circle c1(5.1), c2(12.3);
    cout << "원1의 반지름 : " << c1.getRadius() << endl;
    cout << "원1의 면적 : " << c1.getArea() << endl;
    cout << "원2의 반지름 : " << c2.getRadius() << endl;
    cout << "원2의 면적 : " << c2.getArea() << endl;

    if (c1 >= c2)
        cout << "객체 c1이 c2보다 크다." << endl;
    else
        cout << "객체 c2가 c1보다 크다." << endl;
    return 0;
}
```

```
원1 의 반지름: 5.1
원1 의 면적: 81.7104
원2 의 반지름: 12.3
원2 의 면적: 475.278
객체 c2가 c1보다 크다.
```