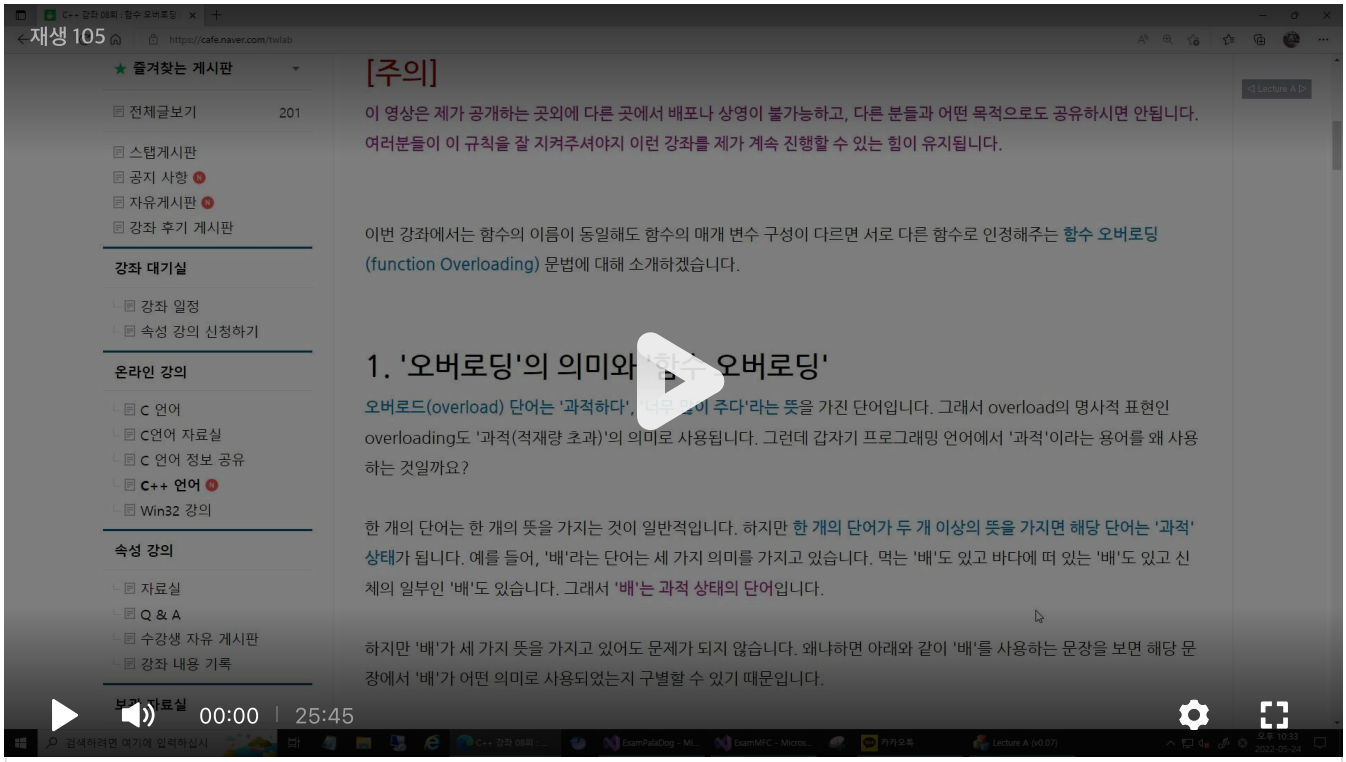
C++ 강좌 08회 : 함수 오버로딩

김성엽 카페매니저 + 구독 1:1 채팅
2022.05.24. 02:49 조회 273

[주의]

이 영상은 제가 공개하는 곳외에 다른 곳에서 배포나 상영이 불가능하고, 다른 분들과 어떤 목적으로도 공유하시면 안됩니다. 여러분들이 이 규칙을 잘 지켜주셔야지 이런 강좌를 제가 계속 진행할 수 있는 힘이 유지됩니다.

이번 강좌에서는 함수의 이름이 동일해도 함수의 매개 변수 구성이 다르면 서로 다른 함수로 인정해주는 함수 오버로딩 (function overloading) 문법에 대해 소개하겠습니다.



C++ 강좌 8회차 : 함수 오버로딩

1. '오버로딩'의 의미와 '함수 오버로딩'

오버로드(overload) 단어는 '과적하다', '너무 많이 주다'라는 뜻을 가진 단어입니다. 그래서 overload의 명사적 표현인 overloading도 '과적(적재량 초과)'의 의미로 사용됩니다. 그런데 갑자기 프로그래밍 언어에서 '과적'이라는 용어를 왜 사용하는 것일까요?

한 개의 단어는 한 개의 뜻을 가지는 것이 일반적입니다. 하지만 한 개의 단어가 두 개 이상의 뜻은 가지면 해당 단어는 '과적' 상태가 됩니다. 예를 들어, '배'라는 단어는 세 가지 의미를 가지고 있습니다. 먹는 '배'도 있고 바다에 떠 있는 '배'도 있고 신체의 일부인 '배'도 있습니다. 그래서 '배'는 과적 상태의 단어입니다.

하지만 '배'가 세 가지 뜻을 가지고 있어도 문제가 되지 않습니다. 왜냐하면 아래와 같이 '배'를 사용하는 문장을 보면 해당 문장에서 '배'가 어떤 의미로 사용되었는지 구별할 수 있기 때문입니다.

```
1. 아침에 시장에서 산 배가 맛있다.   (먹는 '배')
2. 어제부터 배가 아프다.  (신체 일부인 '배')
3. 배를 타고 제주도로 이동했다.  (바다에 떠 있는 '배')
```

C 언어는 함수의 이름으로 사용된 단어에 대해 오버로딩은 허용하지 않습니다. 예를 들어, 소스에 Add 함수가 있는데 다시 Add 함수를 추가하면 오류처리 됩니다. 이것은 C 언어 컴파일러가 C++ 언어 컴파일러 능력이 떨어져서 그런 것이 아니라 C 언어가 운영체제를 만들기 위한 언어이기 때문입니다. 운영체제(OS) 또는 운영환경(OE)을 설계하는 개발자 입장에서는 응용 프로그램 개발을 위해 제공하는 함수(API)를 만들거나 확장 인프라(ex 드라이버)를 설계할 때 함수 이름이 고유해야지 가능한 기술들이 많기 때문입니다.

하지만 C++ 언어는 운영체제를 만들기 위한 언어로 개발된 것이 아니기 때문에 함수 이름은 고유하게 가지는 것보다 이름 중복(오버로딩)은 허용해서 개발자가 함수의 이름은 지으면서 받는 스트레스를 줄여주는 방향은 선택합니다. 왜냐하면 좀 큰 규모의 프로그램을 만들게 되면 수 천개의 함수로 구성되는 경우가 많은데 이 수 천개의 함수 이름을 특별하면서도 고유하게 짓는 작업은 인내와 고통이 따르는 작업입니다. 그중에서도 동일한 기능(행위)를 하는 작업인데 함수 이름을 다르게 지으려면 억지로 이름을 구성하게 되어 소스의 가독성을 떨어트리거나 함수의 이름이 성의없다는 지적을 받기도 합니다.

예를 들어, 숫자를 더하는데 정숫값을 두 개 더하는 함수와 세 개 더하는 함수를 만들어야 한다면 이름을 어떻게 정해야 할까요? 이름을 억지로 다르게 주려고 int Add(int a, int b), int Sum(int a, int b, int c)라고 하는 사람도 있을 것이고 그냥 편하게 int Add2(int a, int b), int Add3(int a, int b, int c)처럼 함수 이름을 짓는 사람도 있을 것입니다. 전자의 경우에는 더한다는 의미를 Add와 Sum로 나누어 소스의 가독성을 떨어트릴수 있고 후자는 함수 이름 뒤에 성의없이 숫자를 붙였다고 비난을 받을 수도 있습니다. 여러분들은 이런 상황에 어떤 이름을 사용하겠습니까?

그래서 C++ 언어는 개발자가 함수 이름을 지을 때 이런 불필요한 스트레스를 받지 않도록 유사한 의미를 가진 또는 유사한 행위를 하는 함수라면 이름은 동일하게 사용할 수 있도록 '함수 오버로딩(function overloading)' 문법은 제공합니다. 따라서 위와 같은 상황에서 함수 이름을 Add로 통일해서 int Add(int a, int b), int Add(int a, int b, int c)라고 사용할 수 있습니다.

C++ 언어에서 이것이 가능한 이유는 C 언어는 함수를 이름으로 구별하지만 C++ 언어는 함수의 이름과 함수의 매개 변수를 조합해서 구별하기 때문입니다. 예를 들어, 아래와 같이 함수의 이름이 같아도 매개 변수의 개수가 다르면 서로 다른 함수로 처리합니다.

```
int Add(int a, int b)  // 매개 변수가 두 개인 함수
{
    return a + b;
}

int Add(int a, int b, int c)  // 매개 변수가 세 개인 함수
{
    return a + b + c;
}
```

이때 Add 함수의 이름이 같아도 사용에 문제가 없는 이유는 **C++ 언어가 함수 호출 구조를 만들 때, 함수 이름뿐만 아니라 해당 함수의 인자 개수도 함께 체크**하기 때문입니다. 즉, 아래와 같이 함수 호출 코드를 사용하면 첫 번째 줄에서는 Add 함수가 두 개의 인자를 사용했기 때문에 int Add(int a, int b) 함수가 호출되고 두 번째 줄에서는 Add 함수가 세 개의 인자를 사용했기 때문에 int Add(int a, int b, int c) 함수를 호출하는 것입니다.

```
int result = Add(2, 3);  // 인자가 두 개이기 때문에 int Add(int a, int b) 함수 호출
int temp = Add(2, 3, 4);  // 인자가 세 개이기 때문에 int Add(int a, int b, int c) 함수 호출
```

앞에서 '배' 단어가 여러 개의 뜻을 가지고 있어도 문장에서 의미를 파악하면 구별이 가능했던 것처럼 **C++ 언어도 Add 함수를 사용한 문장에서 함수가 어떻게 사용되었는지를 체크해서 의미를 구별**하는 것입니다.

그리고 아래와 같이 **함수의 이름이 같고 매개 변수의 개수가 같아도 매개 변수의 자료형이 다르면 서로 다른 함수**로 처리합니다.

```
int Add(int a, int b)  // 두 개의 정숫값을 더하는 함수
{
    return a + b;
}

double Add(double a, double b)  // 두 개의 실숫값을 더하는 함수
{
    return a + b;
}
```

이 경우에는 아래와 같이 함수 호출 코드를 사용했을 때 **함수에 사용된 인자의 개수뿐만 아니라 인자의 자료형도 체크**하기 때문에 문제가 없습니다. 첫 번째 줄에서는 Add 함수의 인자가 2, 3처럼 정수 상수가 사용되었기 때문에 int Add(int a, int b) 함수가 호출됩니다. 그리고 두 번째 줄에서는 Add 함수의 인자가 2.1, 3.5처럼 실수 상수가 사용되었기 때문에 double Add(double a, double b) 함수가 호출되어 구별되는 것입니다.

```
int result = Add(2, 3);  //  인자가 정수라서 int Add(int a, int b) 함수 호출
double temp = Add(2.1, 3.5);  // 인자가 실수라서 double Add(double a, double b) 함수 호출
```

하지만 아래와 같이 **함수의 반환 자료형만 다른 경우에는 함수 오버로딩이 적용되지 않습니다.**

```
int Add(int a, int b)
{
    return a + b;
}

double Add(int a, int b)  // 반환 자료형만 double로 다른 경우, 오류로 처리 됩니다!!
{
    return (double)a + b;
}
```

2. 함수 오버로딩의 모호성

함수 오버로딩은 좋은 기술이지만 생각보다 개발자들이 많이 사용하지는 않습니다. 왜냐하면 함수 오버로딩 문법이 모호성 문제를 가지고 있기 때문입니다. 예를 들어, 아래와 같이 함수 오버로딩을 사용하여 Add 함수를 두 개 정의하고 main 함수에서 Add(2, 3.5);라고 호출하면 컴파일러 입장에서는 **사용된 인자가 두 개로 동일한데 한 개는 정수이고 한 개는 실수라서 int Add(int a, int b) 함수를 호출해야할 지 아니면 double Add(double a, double b) 함수를 호출해야할 지 결정할 수 없기 때문**입니다. 따라서 아래의 코드는 int result = Add(2, 3.5); 코드에서 오류가 발생하게 됩니다.

```
int Add(int a, int b)  // 두 개의 정숫값을 더하는 함수
{
    return a + b;
}

double Add(double a, double b)  // 두 개의 실숫값을 더하는 함수
{
    return a + b;
}

int main()
{
    int result = Add(2, 3.5);  // 오버로딩의 모호성으로 인한 문법 오류 발생!
    return 0;
}
```

그리고 int result = Add(2, 3.5); 코드에서 반환 값이 int 변수인 result에 저장되기 때문에 컴파일러가 int Add(int a, int b) 함수를 선택해주는 것이 가능하다고 생각하는 분들도 있겠지만 함수 오버로딩에서는 반환 자료형은 체크 대상에 포함되지 않습니다.

사실 위와 같은 모호성은 컴파일러가 오류처리를 해주기 때문에 문제가 되지는 않습니다. 오류가 발생한 코드에 가서 상황을 파악해 인자의 자료형만 변경해주면 되기 때문입니다. 하지만 아래와 같은 상황이 정말 문제입니다.

Div 함수는 나눗셈을 하는 함수이고 한 개는 정숫값을 나눗셈하여 몫을 반환하고 한 개는 실숫값을 나눗셈하여 결과를 반환합니다. main 함수의 첫 번째 Div 함수는 두 개의 인자가 2.0, 4.0으로 실수 상숫값이기 때문에 double Div(double a, double b) 함수가 호출되어 result 변수에 정상적으로 0.5 값이 저장됩니다.

하지만 수학 시간에 값은 적은 때 2.0, 4.0처럼 소숫점 이하 값이 0인 숫자는 .0은 생략하고 2, 4로 적었던 경우가 많은 것입니다. 그래서 그 습관 때문에 프로그래밍에서도 자료형을 무시하고 2.0, 4.0을 2, 4라고 적는 사람이 있다는 것입니다. 즉, main 함수의 두 번째 Div 함수에서도 2.0, 4.0으로 적어야 하는데 개발자가 실수로 두 개의 인자를 2, 4로 적었다면 컴파일러는 개발자가 정수 상숫값을 적었기 때문에 반환 자료형과 상관없이 int Div(int a, int b) 함수가 호출되게 구조를 만든 것입니다. 따라서 2/4의 몫은 0이기 때문에 Div 함수는 0을 반환하고 이 값은 temp에 저장됩니다. 그래서 temp 값은 최종적으로 0 값이 저장됩니다.

```
int Div(int a, int b)  // a / b
{
    return a / b;
}

double Div(double a, double b)  // a / b
{
    return a / b;
}

int main()
{
    double result = Div(2.0, 4.0);  // result에 0.5 값이 저장됨
    double temp = Div(2, 4);  // temp에 0 값이 저장됨
    return 0;
}
```

이처럼 단순한 숫자 생략 실수가 문법 오류 없이 컴파일이 되고 완전히 다른 결과를 가져오게 되는 문제가 발생하면 이 문제를 찾기가 쉽지 않습니다. 그래서 초보 개발자나 주의력이 부족한 개발자와 함께 개발할 때 함수 오버로딩을 사용했다가 위와 같은 문제가 발생하면 이 문제를 해결하기 위해 엄청난 시간을 사용해야 할 것입니다. 따라서 이런 경험을 몇 번하고 나면 함수 오버로딩이 아무리 좋은 기술이라고해도 망설여지게 될 것입니다.

3. 함수 오버로딩의 호환성

C 언어는 중복된 이름의 함수를 허용하지 않고 C++ 언어는 중복된 이름의 함수를 허용하기 때문에 C++ 언어는 C 언어로 만든 함수를 호출할 수 있지만, C 언어는 C++ 언어로 만든 함수를 호출할 수 없습니다. 그래서 C, C++ 언어 모두 사용해야할 함수를 만들 때는 C++ 형식이 아닌 C 형식으로 함수를 만들게 됩니다.

만약, C++ 언어로 만든 함수를 C 언어에서 호출하게 하고 싶다면 아래와 같이 extern "C" 문법을 사용해서 함수를 정의하면 됩니다. extern "C" 구문에 사용된 코드는 C++ 언어가 아닌 C 언어 문법이 적용됩니다.

```
extern "C"
{
    int Div(int a, int b)
    {
        return a / b;
    }
}
```

따라서 아래와 같이 extern "C" 구문에서 함수 오버로딩 문법을 사용하면 C 언어가 오버로딩을 제공하지 않기 때문에 컴파일러가 실수 형식을 사용하는 두 번째 Div 함수를 컴파일할 때 오류 처리합니다.

```
extern "C"
{
    int Div(int a, int b)
    {
        return a / b;
    }

    double Div(double a, double b)  // 오버로딩이 적용되지 않기 때문에 컴파일 오류 발생
    {
        return a / b;
    }
}
```

클린봇이 악성 댓글을 감지합니다. 설정

댓글 등록순 최신순 관심글 댓글 알림

카일 3
함수 오버로딩에 의해 발생할 수 있는 문제에 대해 생각해 보는 시간이었습니다. 강의 잘 들었습니다.
2022.11.25. 20:11 답글쓰기

김성엽 M 작성자

수고하셨습니다
2022.11.25. 21:51 답글쓰기

dh221009
댓글을 남겨보세요
등록