

C++ 강좌 02회 : 포인터 표현을 감추기 위한 노력 - this 포인터



김성엽 카페매니저



+ 구독

1:1 채팅

2022.05.06. 10:33 조회 581



댓글 28

URL 복사



[주의]

이 영상은 제가 공개하는 곳 외에 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.

아래의 예제 코드는 [01회차] 강좌에서 소개했던 좋은 코드의 일반적인 형태입니다. 하지만 함수, 구조체 그리고 포인터 문법이 사용되는 코드라서 초보자들이 쉽게 이해하기도 힘들고 이 코드를 프로그램에서 활용하기도 어렵습니다. 그래서 이번 강좌에서는 이 어려운 코드를 초보자들이 쉽게 이해하고 사용할 수 있도록 C++ 언어의 문법 표현이 어떻게 변경되었는지 소개하겠습니다.

```

#include <stdio.h> // printf, scanf 함수를 사용하기 위해!


// 함수가 사용할 매개 변수를 새로운 자료형으로 선언
// 따라서 이제 함수의 매개 변수에 변화가 생기면 여기에
// 변화가 생기고 함수의 매개 변수 형식에는 변화가 생기지 않는다.
struct ValueList
{
    int a, b, c;
};


void ModifyValue(int *p_value, const char *p_var_name)
{
    if (*p_value < 0)
        printf("%s변수의 값을 %d에서 %d로 변경함\n", p_var_name, -*p_value, *p_value = -*p_value);
}


void MultiplyValue(struct ValueList *p_values)
{
    // a, b 또는 c 변수가 음수라면 양수로 변경한다.
    ModifyValue(&p_values->a, "a");
    ModifyValue(&p_values->b, "b");
    ModifyValue(&p_values->c, "c");

    // 전달된 세 수의 곱셈 결과를 출력한다.
    printf("%d * %d * %d = %d\n", p_values->a, p_values->b, p_values->c,
        p_values->a * p_values->b * p_values->c);
}

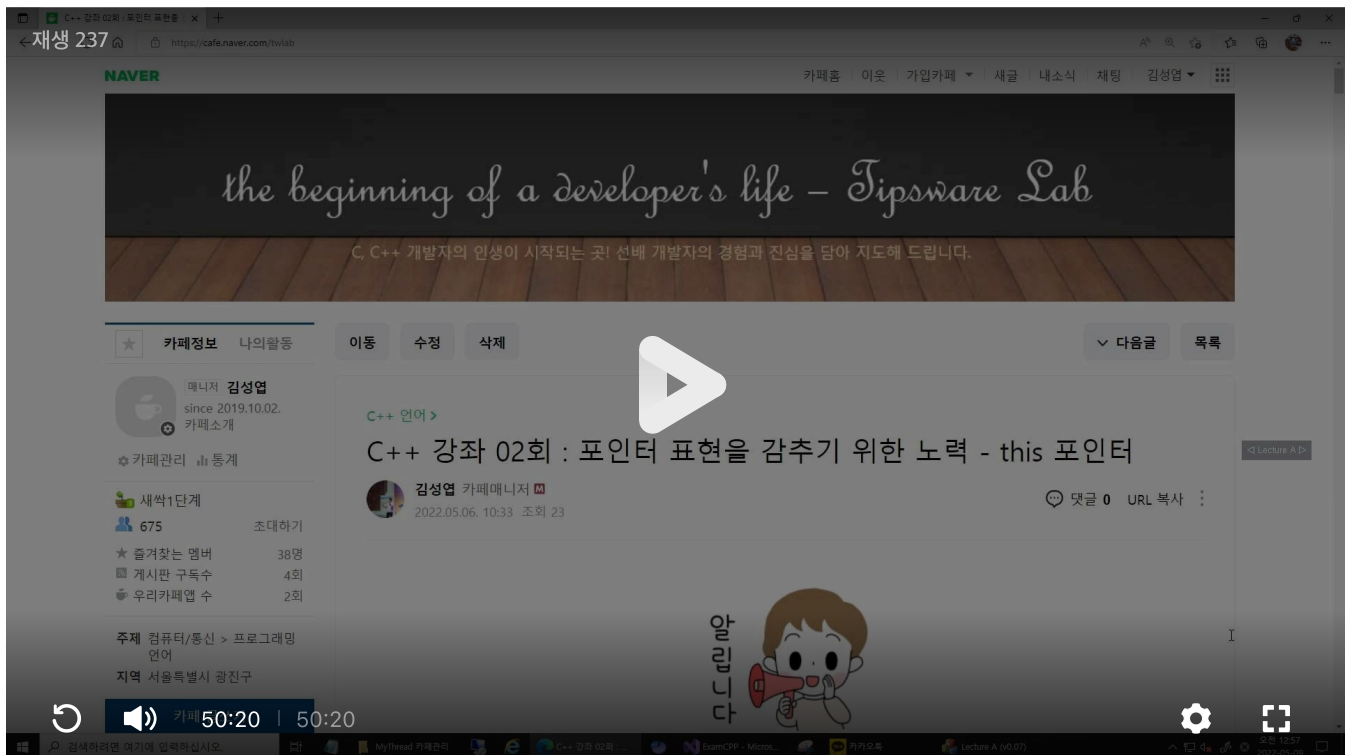

int main()
{
    struct ValueList values;

    printf("곱셈할 수를 입력하세요: ");
    // 세 개의 정숫값을 입력받는다.
    scanf_s("%d %d %d", &values.a, &values.b, &values.c);
    MultiplyValue(&values); // 입력받은 변수값 사용 가능

    struct ValueList temp = { 2, -6, -3 }; // 상숫값 사용 가능
    MultiplyValue(&temp);

    return 0;
}

```



02회 포인터 표현을 감추기 위한 노력 - this 포인터

1. 자료형의 강화

자료형은 프로그래밍언어에서 **개발자가 메모리를 쉽게 사용할 수 있게 도와주는 문법**입니다. 그리고 자료형은 개발자가 **메모리를 사용함에 있어 실수나 오류가 없는지 체크**해주는 기본적인면서도 중요한 문법입니다. 그래서 자료형이라는 개념이 프로그래밍 언어에 도입되면서 프로그래밍 언어가 쉬워지고 개발자의 실수가 줄어들기 시작한 것입니다.

물론 자료형이라는 개념은 프로그래밍 언어가 제공하는 논리적인 개념(규칙)이라서 C 언어로 작성된 소스가 기계어로 번역될 때 특정 명령문을 구성하는 항목 값으로 번역은 되지만 자료형 자체가 컴퓨터의 명령문으로 번역되지 않습니다. 즉, 프로그램을 구성하는 명령문들은 함수나 연산자 중심으로 구성되어 있고 해당 명령문을 구성할 때 크기를 지정하는 값으로만 자료형이 사용됩니다.

하지만 기계어에서 제공하지 않는 개념일지라도 자료형을 사용함으로써 더 편하게 소스를 작성할 수 있고 개발자의 실수도 방지할 수 있기 때문에 프로그래밍 언어가 발전하면서 자료형 개념도 함께 발전하고 확대 적용되고 있습니다. 즉, **자료형을 강화한다는 뜻은 개발자가 프로그래밍을 더 쉽게 할 수 있도록 도와주는 문법이 추가된다는 뜻**입니다.

그래서 C++ 언어도 C 언어의 포인터 표현과 구조체 표현을 좀더 단순화시켜 개발자가 좀더 쉬운 표현으로 사용할 수 있도록 자료형을 강화하는 선택을 합니다. 먼저 구조체 사용을 편하게 할 수 있도록 C 언어에서 구조체로 만든 자료형을 사용할 때 struct 키워드를 반드시 적어야 했던 문법을 생략 가능하게 변경했습니다. 예를 들어, ValueList라는 구조체를 선언했다면 이 구조체를 사용해서 변수를 선언할 때 **C 언어는 struct ValueList values;처럼 사용하지만 C++ 언어는 ValueList values;라고 사용**해도 됩니다.

그리고 C 언어에서는 구조체가 데이터를 그룹지어 새로운 자료형으로 만드는 문법으로 한정되어 있었는데 C++ 언어에서는 구조체에 함수를 추가할 수 있도록 변경되었습니다. 이것은 구조체를 선언하면 구조체를 사용하는 함수들이 정의되기 마련인데, 이 두 가지 표현을 분리해서 사용하면 포인터 문법이 많이 사용되기 때문에 포인터 표현을 줄이기 위해 두 표현을 하나로 묶는 선택을 한 것입니다.

예를 들어, C++ 언어는 자료형을 강화하는 첫 단계로 아래의 예제처럼 ValueList 구조체를 선언하면서 ValueList 구조체를 사용하는 MultiplyValue 함수를 포함시키는 표현 방법을 제공합니다. 이렇게 문법이 사용되면 이제 MultiplyValue 함수는 ValueList 구조체에 포함된 함수이기 때문에 main 함수에서 사용할 때 바로 호출을 할 수 없고 ValueList 자료형으로 선언된 변수를 사용해서만 호출이 가능합니다. 그래서 `values.MultiplyValue(&values);` 또는 `temp.MultiplyValue(&temp);` 처럼 사용됩니다.

```

#include <stdio.h> // printf, scanf 함수를 사용하기 위해!

void ModifyValue(int *p_value, const char *p_var_name)
{
    if (*p_value < 0)
        printf("%s변수의 값을 %d에서 %d로 변경함\n", p_var_name, -*p_value, *p_value = -*p_value);
}

struct ValueList
{
    int a, b, c;

    void MultiplyValue(ValueList *p_values)
    {
        // a, b 또는 c 변수가 음수라면 양수로 변경한다.
        ModifyValue(&p_values->a, "a");
        ModifyValue(&p_values->b, "b");
        ModifyValue(&p_values->c, "c");

        // 전달된 세 수의 곱셈 결과를 출력한다.
        printf("%d * %d * %d = %d\n", p_values->a, p_values->b, p_values->c,
            p_values->a * p_values->b * p_values->c);
    }
};

int main()
{
    ValueList values;

    printf("곱셈할 수를 입력하세요: ");
    // 세 개의 정숫값을 입력받는다.
    scanf_s("%d %d %d", &values.a, &values.b, &values.c);
    values.MultiplyValue(&values); // 입력받은 변수값 사용 가능

    ValueList temp = { 2, -6, -3 }; // 상숫값 사용 가능
    temp.MultiplyValue(&temp);

    return 0;
}

```

2. 주소 표현의 단순화

C++ 언어는 자료형 강화를 통해 구조체에 변수뿐만 아니라 함수도 항목으로 가질수 있습니다. 그래서 구조체에 선언된 함수는 아래와 같이 해당 구조체로 선언된 변수를 사용해서 호출해야 합니다. 그런데 아래의 코드를 보면 **MultiplyValue 함수를 호출하면서 values라는 변수가 중복해서 사용된 것**을 볼 수 있습니다.

```
values.MultiplyValue(&values); // 입력받은 변수값 사용 가능
```

이 표현을 컴파일러 입장에서 보면 `values.MultiplyValue` 표현은 `MultiplyValue` 함수가 `values` 변수에 포함되어 있다는 뜻입니다. 그런데 `MultiplyValue` 함수가 `values` 변수에 포함되어 있다는 뜻은 이 함수가 `values` 변수 내부에 있는 데이터를 사용하기 위해 포함되어 있다는 뜻입니다. 즉, `MultiplyValue` 함수가 `values` 변수(내부 데이터)를 사용하겠다는 뜻이 포함되어 있습니다.

그래서 `values.MultiplyValue` 표현은 `MultiplyValue(&values)` 표현을 포함하는 의미이기 때문에 굳이 매개 변수로 `&values`를 적지 않더라도 C++ 컴파일러가 대신 처리하게 만들 수 있습니다. 즉, C++ 컴파일러가 `values.MultiplyValue` 코드를 기계어로 번역할 때 `MultiplyValue` 함수를 호출하면서 매개 변수로 `values` 변수의 주소를 전달하는 형식으로 코드를 구성하게 만들 수 있다는 뜻이고 실제로 이렇게 구현되어 있습니다. 따라서 `values.MultiplyValue(&values);`와 같은 형식의 코드는 아래와 같이 `&values`를 사용할 필요가 없습니다.

```
values.MultiplyValue(); // MultiplyValue 함수가 호출되면서 &values 값이 전달
```

3. this 포인터

2번 항목에서 `values.MultiplyValue(&values);` 코드를 `values.MultiplyValue();` 이렇게 사용해도 된다고 했는데 막상 이렇게 사용하면 오류가 발생할 것입니다. 이것은 문법 기준으로 볼 때 함수를 호출하는 쪽에서는 매개 변수를 사용하지 않았는데 `ValueList` 구조체에 선언된 `MultiplyValue` 함수의 원형은 `void MultiplyValue(ValueList *p_values)`라고 매개 변수가 선언되어 있기 때문입니다.

그래서 문법적으로 `&values`를 생략하고 싶다면 `ValueList` 구조체에 선언된 `MultiplyValue` 함수에 사용된 `ValueList *p_values` 매개 변수 선언도 같이 생략할 수 있어야 합니다. 하지만 이것은 쉬운일이 아닙니다.

`values.MultiplyValue(&values);` 코드에서 `&values`를 생략하는 것은 `values.`을 사용해서 유추할 수 있지만 `ValueList` 구조체의 `void MultiplyValue(ValueList *p_values)`에서 `ValueList *p_values`를 생략하게 되면 `p_values`라는 변수 이름이 사라져 버려서 어떤 변수를 사용하는지 알 수 없기 때문입니다. 즉, 어떤 정보를 생략하려면 생략해도 그 정보가 있음을 유추할 수 있어야 하는데 변수명은 개발자가 변경 가능하기 때문에 유추한다는 것이 사실상 어렵습니다.

그래서 이렇게 생략된 변수의 이름을 유추하게 하려면 개발자와 컴파일러가 이름에 대한 약속을 해야하기 때문에 C++에 `this`라는 이름의 키워드가 추가된 것입니다. 즉, 지금과 같은 상황으로 생략된 포인터 변수의 이름이 `this`입니다. 따라서 아래와 같이 C++ 구조체에 함수를 선언해서 사용하게 되면 이 함수에는 자동으로 `this`라는 포인터가 생략되어 있기 때문에 별도로 선언하지 않아도 이 함수에서 사용이 가능합니다. 물론 이 포인터의 주소는 해당 구조체의 주소를 사용해야 하는 것으로 약속되어 있기 때문에 개발자가 임의로 변경하지 못하게 `const` 키워드를 함께 사용해서 선언되어 있습니다.

```

#include <stdio.h> // printf, scanf 함수를 사용하기 위해!

void ModifyValue(int *p_value, const char *p_var_name)
{
    if (*p_value < 0)
        printf("%s변수의 값을 %d에서 %d로 변경함\n", p_var_name, -*p_value, *p_value = -*p_value);
}

struct ValueList
{
    int a, b, c;

    void MultiplyValue() // ValueList *const this; 생략 됨
    {
        // a, b 또는 c 변수가 음수라면 양수로 변경한다.
        ModifyValue(&this->a, "a");
        ModifyValue(&this->b, "b");
        ModifyValue(&this->c, "c");

        // 전달된 세 수의 곱셈 결과를 출력한다.
        printf("%d * %d * %d = %d\n", this->a, this->b, this->c,
            this->a * this->b * this->c);
    }
};

int main()
{
    ValueList values;

    printf("곱셈할 수를 입력하세요: ");
    // 세 개의 정숫값을 입력받는다.
    scanf_s("%d %d %d", &values.a, &values.b, &values.c);
    values.MultiplyValue(); // 입력받은 변수값 사용 가능

    ValueList temp = { 2, -6, -3 }; // 상숫값 사용 가능
    temp.MultiplyValue();

    return 0;
}

```

그런데 위 코드를 초보자들이 사용해야 하는데 this 포인터가 위와 같이 사용되면 이 개념을 이해하기가 더 어려울 것입니다. 즉, '존재하지 않는 변수를 사용할 수 있다'라는 의미를 초보자들이 쉽게 받아들이기는 어려울 것입니다. 그래서 C++ 언어는 **this 포인터가 생략되어 선언되었으니 사용할 때도 생략해서 사용할 수 있는 문법을 제공합니다.**

그래서 또 다른 규칙(문법)이 추가되었습니다. **구조체 내부에 선언된 함수에서 선언되지 않은 변수가 사용되었다면 컴파일러는 그 변수에 this->를 자동으로 붙여서 한 번더 체크하는 것입니다.** 예를 들어, ValueList 구조체에서 MultiplyValue 함수에 사용된 this-> 표현을 모두 생략하고 아래와 같이 적어도 오류가 발생하지 않는다는 뜻입니다. C 언어 기준에서는 MultiplyValue 함수에 사용된 a, b, c 변수는 선언되지 않은 변수이기 때문에 모두 오류 처리가 되겠지만 C++ 언어에서는

선언되지 않은 변수가 사용되면 해당 변수 앞에 this->를 붙여서 다시 체크하기 때문에 a, b, c라고 사용된 코드는 실제로 this->a, this->b, this->c라고 사용한 것과 동일하여 오류가 발생하지 않습니다.

```
struct ValueList
{
    int a, b, c;

    void MultiplyValue() // ValueList *const this; 생략 됨
    {
        // a, b 또는 c 변수가 음수라면 양수로 변경한다.
        ModifyValue(&a, "a");
        ModifyValue(&b, "b");
        ModifyValue(&c, "c");

        // 전달된 세 수의 곱셈 결과를 출력한다.
        printf("%d * %d * %d = %d\n", a, b, c, a * b * c);
    }
};
```

하지만 이런 관계를 초보자들에게 설명하기는 어렵겠죠? 그래서 가르치는 분들은 이 내용을 쉽게 설명하기 위해 선의의 거짓 말을 아래와 같이 초보자들에게 하는 것입니다. 구조체 선언된 **a, b, c 변수는 '멤버 변수'라고 부르고 MultiplyValue 함수는 '멤버 함수'라고 합니다.** 즉, 전역 변수 같은 개념이 초보자들에게 편하게 설명할 수 있는 개념이기 때문에 내부적으로는 포인터가 사용되어 구현되었다고 해도 문법적으로는 포인터가 노출이 되지 않기 때문에 포인터에 대한 이야기는 숨기고 그냥 전역 변수인 것처럼 설명하고 넘어가는 것입니다. 그래서 C++를 사용하는 개발자들중에 this 포인터에 대해 제대로 이해하는 사람이 많지 않습니다.

**" 구조체의 멤버 변수는 구조체에 선언된 전역 변수 같은 개념이라서
멤버 함수에서는 별도의 선언이나 키워드 사용 없이 바로 사용이 가능합니다. "**

그래서 전체적으로 소스가 아래와 같이 변경되어 이 글의 시작에서 보여드렸던 포인터 문법이 많이 감춰지게 됩니다.


```

#include <stdio.h> // printf, scanf 함수를 사용하기 위해!

void ModifyValue(int *p_value, const char *p_var_name)
{
    if (*p_value < 0)
        printf("%s변수의 값을 %d에서 %d로 변경함\n", p_var_name, -*p_value, *p_value = -*p_value);
}

struct ValueList
{
    int a, b, c;

    void MultiplyValue() // ValueList *const this; 생략 됨
    {
        // a, b 또는 c 변수가 음수라면 양수로 변경한다.
        ModifyValue(&a, "a");
        ModifyValue(&b, "b");
        ModifyValue(&c, "c");

        // 전달된 세 수의 곱셈 결과를 출력한다.
        printf("%d * %d * %d = %d\n", a, b, c, a * b * c);
    }
};

int main()
{
    ValueList values;

    printf("곱셈할 수를 입력하세요: ");
    // 세 개의 정숫값을 입력받는다.
    scanf_s("%d %d %d", &values.a, &values.b, &values.c);
    values.MultiplyValue(); // 입력받은 변수값 사용 가능

    ValueList temp = { 2, -6, -3 }; // 상숫값 사용 가능
    temp.MultiplyValue();

    return 0;
}

```

그런데 위 예제에서 함께 사용된 ModifyValue 함수의 내용이 초보자들이 보기에는 너무 어려울 것입니다. 그래서 이 코드를 초보자들이 이해할 수 있는 수준으로 변경하기 위해 ModifyValue 함수를 ValueList 구조체의 멤버 함수로 변경하겠습니다. 즉, 아래와 같이 코드를 구성하면 포인터 표현이 사라져서 아무래도 초보자들이 보기에는 더 쉬운것처럼 느낄 것입니다.

```

#include <stdio.h> // printf, scanf 함수를 사용하기 위해!

struct ValueList
{
    int a, b, c;

    void MultiplyValue()
    {
        // a, b 또는 c 변수가 음수라면 양수로 변경한다.
        ModifyValue();

        // 전달된 세 수의 곱셈 결과를 출력한다.
        printf("%d * %d * %d = %d\n", a, b, c, a * b * c);
    }

    void ModifyValue()
    {
        if (a < 0) printf("a 변수의 값을 %d에서 %d로 변경함\n", -a, a = -a);
        if (b < 0) printf("b 변수의 값을 %d에서 %d로 변경함\n", -b, b = -b);
        if (c < 0) printf("c 변수의 값을 %d에서 %d로 변경함\n", -c, c = -c);
    }
};

int main()
{
    ValueList values;

    printf("곱셈할 수를 입력하세요: ");
    // 세 개의 정숫값을 입력받는다.
    scanf_s("%d %d %d", &values.a, &values.b, &values.c);
    values.MultiplyValue(); // 입력받은 변수값 사용 가능

    ValueList temp = { 2, -6, -3 }; // 상숫값 사용 가능
    temp.MultiplyValue();

    return 0;
}

```

위 예제는 아래와 같이 출력됩니다.

```

곱셈할 수를 입력하세요: 5 -1 -3
b 변수의 값을 -1에서 1로 변경함
c 변수의 값을 -3에서 3로 변경함
5 * 1 * 3 = 15
b 변수의 값을 -6에서 6로 변경함
c 변수의 값을 -3에서 3로 변경함
2 * 6 * 3 = 36

```

이 창을 닫으려면 아무 키나 누르세요...

위 코드에서 ModifyValue 함수에 this 포인터를 생략하지 않고 다 적어보면 다음과 같습니다. this-> 코드를 생략했을 때와 비교해보시면 알겠지만 초보자들이 소스를 본다고 생각한다면 그 난이도 차이는 클 것입니다.

```
struct ValueList
{
    int a, b, c;

    void MultiplyValue()
    {
        // a, b 또는 c 변수가 음수라면 양수로 변경한다.
        ModifyValue();

        // 전달된 세 수의 곱셈 결과를 출력한다.
        printf("%d * %d * %d = %d\n", a, b, c, a * b * c);
    }

    void ModifyValue() // ValueList *const this; 생략 됨
    {
        if (this->a < 0) printf("a 변수의 값을 %d에서 %d로 변경함\n", -this->a, this->a = -this->a);
        if (this->b < 0) printf("b 변수의 값을 %d에서 %d로 변경함\n", -this->b, this->b = -this->b);
        if (this->c < 0) printf("c 변수의 값을 %d에서 %d로 변경함\n", -this->c, this->c = -this->c);
    }
};
```

4. this 포인터를 사용할 때 주의해야 할 점과 해결책

this 포인터는 선언되지 않은 변수에 사용되기 때문에 개발자가 실수로 멤버 변수와 동일한 변수를 지역 변수로 사용하게 되면 개발자의 의도와 다르게 동작할 수 있습니다. 예를 들어, 아래와 같이 ModifyValue 함수에 지역 변수 a를 선언하게 되면 b와 c에는 this->가 자동으로 붙어서 처리되지만 a 변수에는 this->가 적용되지 않습니다. 즉, ModifyValue 함수에 지역 변수 a가 선언되었기 때문에 a라고 사용하면 멤버 변수 a가 아닌 지역 변수 a가 사용되는 것입니다.

```

struct ValueList
{
    int a, b, c; // 멤버 변수 선언

    void MultiplyValue()
    {
        // a, b 또는 c 변수가 음수라면 양수로 변경한다.
        ModifyValue();

        // 전달된 세 수의 곱셈 결과를 출력한다.
        printf("%d * %d * %d = %d\n", a, b, c, a * b * c);
    }

    void ModifyValue()
    {
        int a = 0; // 지역 변수 선언

        if (a < 0) printf("a 변수의 값을 %d에서 %d로 변경함\n", -a, a = -a);
        if (b < 0) printf("b 변수의 값을 %d에서 %d로 변경함\n", -b, b = -b);
        if (c < 0) printf("c 변수의 값을 %d에서 %d로 변경함\n", -c, c = -c);
    }
};

```

따라서 위와 같은 코드에서 ModifyValue 함수에 사용된 a 변수를 지역 변수 a가 아닌 **멤버 변수 a**를 사용해야 한다면 b와 c 변수는 **this->** 코드를 생략해도 되지만 a 변수에 대해서는 생략하지 못하고 아래와 같이 **this->**를 붙여서 사용해야 합니다.

```

struct ValueList
{
    int a, b, c; // 멤버 변수 선언

    void MultiplyValue()
    {
        // a, b 또는 c 변수가 음수라면 양수로 변경한다.
        ModifyValue();

        // 전달된 세 수의 곱셈 결과를 출력한다.
        printf("%d * %d * %d = %d\n", a, b, c, a * b * c);
    }

    void ModifyValue()
    {
        int a = 0; // 지역 변수 선언

        // 지역 변수 a가 아닌 멤버 변수 a를 사용하고 싶다면 this-> 붙여서 사용하면 된다.
        if (this->a < 0) printf("a 변수의 값을 %d에서 %d로 변경함\n", -this->a, this->a = -this->a);
        if (b < 0) printf("b 변수의 값을 %d에서 %d로 변경함\n", -b, b = -b);
        if (c < 0) printf("c 변수의 값을 %d에서 %d로 변경함\n", -c, c = -c);
    }
};

```

의도했던 하지 않았던 위와 같은 상황이 발생하면 초보자들을 위해 숨겼던 this-> 표현이 노출되면서 초보자들은 다시 어려워진 개념과 코드를 다루어야 합니다. 그래서 C++ 언어는 C 언어와 달리 **표기법을 더 강조**하는 것입니다. '표기법'이라는 것은 C++ 언어의 문법은 아니지만 관습적으로 변수나 함수를 선언할 때 개발자들 간에 약속된 표현을 사용하는 것입니다. **프로그래밍 언어에서 표기법은 소스의 가독성(읽기 편함 정도)을 높이기 위해 적절하게 사용하는 것이 좋다**고 되어 있기 때문에 결과적으로는 표기법을 사용하는 것이 좋긴합니다.

예를 들어, 구조체에 멤버 변수를 선언할 때는 다른 상황에서 선언된 변수 이름과 구분하기 위해서 멤버 변수 이름 앞에 아래와 같이 'm_' (member)를 붙여서 선언하는 표기법을 권장하는 것입니다. 비슷한 의미로 전역 변수 이름 앞에는 'g_' (global)를 붙여서 선언하는 경우도 많습니다. 표기법은 개발자가 사용하는 개념이기 때문에 사용자가 보는 출력문에는 m_a 같은 표현을 사용하지 않기 때문에 a로 출력합니다.

```
struct ValueList
{
    int m_a, m_b, m_c;    // 멤버 변수 선언

    void MultiplyValue()
    {
        // m_a, m_b 또는 m_c 변수가 음수라면 양수로 변경한다.
        ModifyValue();

        // 전달된 세 수의 곱셈 결과를 출력한다.
        printf("%d * %d * %d = %d\n", m_a, m_b, m_c, m_a * m_b * m_c);
    }

    void ModifyValue()
    {
        if (m_a < 0) printf("a 변수의 값을 %d에서 %d로 변경함\n", -m_a, m_a = -m_a);
        if (m_b < 0) printf("b 변수의 값을 %d에서 %d로 변경함\n", -m_b, m_b = -m_b);
        if (m_c < 0) printf("c 변수의 값을 %d에서 %d로 변경함\n", -m_c, m_c = -m_c);
    }
};
```

이렇게 표기법을 사용하게 되면 특정 함수에 사용된 변수가 지역 변수인지 멤버 변수인지 확실하게 구분할 수 있기 때문에 개발자가 소스를 이해하는데 더 도움은 되지만 코드를 적을 때 타이핑을 더 많이해야 하는 불편함이 있는 건 사실입니다. 하지만 불편함보다 가독성을 높이는 것이 더 장점이 많기 때문에 표기법을 사용해야 한다고 강조합니다.

하지만 이런 교육의 이면에는 이렇게 함으로써 **지역 변수와 멤버 변수의 이름이 절대 중복되는 상황이 나오지 않게해서 C++ 언어를 공부하는 초보자들이 this->와 같이 어려운 코드를 보지 않게 하려는 목적**도 있습니다. 예를 들어, 아래의 예제에서 ModifyValue 함수에 지역 변수로 a 변수를 선언해도 이제 this-> 코드를 추가할 필요가 없어지는 것입니다.

```

struct ValueList
{
    int m_a, m_b, m_c;    // 멤버 변수 선언

    void MultiplyValue()
    {
        // m_a, m_b 또는 m_c 변수가 음수라면 양수로 변경한다.
        ModifyValue();

        // 전달된 세 수의 곱셈 결과를 출력한다.
        printf("%d * %d * %d = %d\n", m_a, m_b, m_c, m_a * m_b * m_c);
    }

    void ModifyValue()
    {
        int a = 0;    // 지역 변수 선언

        // 멤버 변수와 지역 변수는 표기법을 지키면 변수 이름이 중복되지 않아서
        // 생략된 this-> 코드를 적을 확률이 없어진다.
        if (m_a < 0) printf("a 변수의 값을 %d에서 %d로 변경함\n", -m_a, m_a = -m_a);
        if (m_b < 0) printf("b 변수의 값을 %d에서 %d로 변경함\n", -m_b, m_b = -m_b);
        if (m_c < 0) printf("c 변수의 값을 %d에서 %d로 변경함\n", -m_c, m_c = -m_c);
    }
};

```

마지막으로 이번 강좌를 통해 C++ 언어의 기본적인 코드 표현으로 정리된 예제를 보여드리면 다음과 같습니다. 처음 시작할 때 봤던 코드에 비해 포인터 문법이 많이 사라지고 구조체 표현도 줄어 초보자가 읽기 편한 코드가 되었을 것입니다.

```

#include <stdio.h> // printf, scanf 함수를 사용하기 위해!

struct ValueList
{
    int m_a, m_b, m_c;    // 멤버 변수 선언

    void MultiplyValue()
    {
        // m_a, m_b 또는 m_c 변수가 음수라면 양수로 변경한다.
        ModifyValue();

        // 전달된 세 수의 곱셈 결과를 출력한다.
        printf("%d * %d * %d = %d\n", m_a, m_b, m_c, m_a * m_b * m_c);
    }

    void ModifyValue()
    {
        if (m_a < 0) printf("a 변수의 값을 %d에서 %d로 변경함\n", -m_a, m_a = -m_a);
        if (m_b < 0) printf("b 변수의 값을 %d에서 %d로 변경함\n", -m_b, m_b = -m_b);
        if (m_c < 0) printf("c 변수의 값을 %d에서 %d로 변경함\n", -m_c, m_c = -m_c);
    }
};

int main()
{
    ValueList values;

    printf("곱셈할 수를 입력하세요: ");
    // 세 개의 정숫값을 입력받는다.
    scanf_s("%d %d %d", &values.m_a, &values.m_b, &values.m_c);
    values.MultiplyValue(); // 입력받은 변수값 사용 가능


    ValueList temp = { 2, -6, -3 }; // 상숫값 사용 가능
    temp.MultiplyValue();

    return 0;
}

```



어떻게 보면 this 포인터가 소스를 구성할 때 C 언어와 C++ 언어가 추구하는 코드 스타일의 차이를 가장 잘 이해할 수 있는 내용입니다. 따라서 이 강좌를 여러 번 반복해서 보셔서 this 포인터에 대한 이해도를 높이시고 다음 강좌로 넘어가는 것을 추천합니다.

지금은 초보자용 자료이고 구현 설명을 위해 this 포인터를 숨겨야하는 것처럼 이야기하고 처음에는 this 포인터가 보이지 않겠지만 C++ 언어를 사용해서 제대로 프로그래밍을 하게 되면 this 포인터가 자주 등장하게 되기 때문입니다. 결국 포인터 기술은 초보자들에게는 숨길수 있어도 전문가는 사용할 수 밖에 없는 기술입니다.

 클린봇이 악성 댓글을 감지합니다.

 설정

댓글 등록순 최신순 

 관심글 댓글 알림 



조민희 3

C와 C++의 철학 좋은 강의 감사합니다 !!

2022.07.03. 14:26 답글쓰기



김성엽 M 작성자

공부를 시작했군요 ㅎㅎ 파이팅입니다 ㅎㅎ



수고하셨습니다

2022.07.03. 14:27 답글쓰기



조민희 3

values.MultiplyValue()를 호출해서 this 포인터 변수에 &values가 들어가면 ModifyValue() 에서도 this 포인터 변수에 &values가 대입되는 것인가요 ??

2022.07.03. 15:58 답글쓰기



김성엽 M 작성자

네~ 동일한 주소가 넘어갑니다.

2022.07.03. 16:01 답글쓰기



조민희 3

김성엽 그러면 이 상황에서는 this 포인터 변수가 2개가 존재하는 것인가요 ??

2022.07.03. 16:03 답글쓰기



김성엽 M 작성자

조민희 아닙니다 ㅎㅎ MultiplyValue 함수로 전달된 주소는 해당 함수의 this 포인터에 저장되고 그 this 포인터가 그대로 ModifyValue 함수로 전달되고 ModifyValue 함수에서 받아서 사용하는 것입니다. this 포인터는 숨겨진 매개 변수 개념이라 함수가 호출될때마다 생깁니다.

2022.07.03. 17:34 답글쓰기



조민희 3

김성엽 여러 개의 멤버 함수들이 this 포인터 변수를 공용으로 사용하는 개념이네요 !

2022.07.03. 19:26 답글쓰기



김성엽 M 작성자

조민희 ㅎㅎㅎ 공용으로 사용한다는 표현은 오해를 만들수 있습니다. 같은 주소를 사용한다는 개념이지 공용으로 사용하는 개념은 아닙니다. 각 함수에 하나씩 자동으로 추가되어 사용되는 개념입니다~ :)

2022.07.03. 19:34 답글쓰기



조민희 3

김성엽 완벽하게 이해했습니다 ! 친절한 답변 감사드립니다 :)

2022.07.03. 19:42 답글쓰기



김성엽 M 작성자

조민희



화이팅!

2022.07.03. 19:45 답글쓰기



Zermi 3

선생님!

b, c는 멤버 변수로 그대로 사용하고 a를 멤버 변수 나 지역변수로 사용하고 싶으면

멤버 함수(){

int a;

m_a = a;

if(m_a < 0) m_a * -1


```

if(m_b < 0) m_b * -1
if(m_c < 0) m_c * -1
}

```

이렇게 선언해야되는건가요? 이걸 무조건 지역변수 쓰는건데..

```

int a;
if(a != NULL){
    m_a = a;
}
if(m_a < 0) m_a * -1
if(m_b < 0) m_b * -1
if(m_c < 0) m_c * -1
} 대충 이렇게 해야되나..

```

이때 오버로딩 쓰는건가;

아 뭘 모르는지 몰라서 질문도 멍청하네요..

아직 뒤에 안 봐서 현 시점에서 궁금한 점 끄적여 봤습니다!

더 보고 다시 돌아오겠습니다!

2022.09.03. 16:01 답글쓰기



김성엽 ✎ 작성자

ㅎㅎㅎ 왜 그렇게 사용하려고 하는지를 이야기하는 것이 좋지 않을까요? 때로는 생기지 않을 일을 걱정하는 경우도 많습니다.

2022.09.03. 22:50 답글쓰기



티모 3

m_ 를 왜그리 강조했는지 이제 좀 이해가 가네요! 변수에 멤버변수? 그게 뭐야 하면서 제 마음대로 썼었는데..ㅠㅠ 역시 무식하면 용감하다 ㅎㅎ

2022.11.08. 20:51 답글쓰기



김성엽 ✎ 작성자

모든 강조되는 행위에는 이유가 다 있습니다 ㅎㅎ



2022.11.08. 20:50 답글쓰기



황금잉어가물치 3

MultiplyValue함수에서 ValueList *const this에 대해 struct ValueList *const this로 안 적어 놓으신건 struct 구조체 안에 있기에 struct를 안 적으신게 맞는지요? 이해를 잘 못 할 수도 있어서.. 궁금합니다.

```

struct ValueList
{
    int a, b, c;

    void MultiplyValue() // ValueList *const this; 생략 됨
    {
        // a, b 또는 c 변수가 음수라면 양수로 변경한다.
        ModifyValue(&this->a, "a");
        ModifyValue(&this->b, "b");
        ModifyValue(&this->c, "c");

        // 전달된 세 수의 곱셈 결과를 출력한다.
        printf("%d * %d * %d = %d\n", this->a, this->b, this->c,
            this->a * this->b * this->c);
    }
};

```

2022.11.12. 16:55 답글쓰기



김성엽 ✎ 작성자

C언어는 struct 꼭 적어야하고 c++는 생략 가능합니다. 즉, 위치때문이 아니고 c++에서 생략 가능한 거예요 ㅎㅎ

2022.11.12. 17:48 답글쓰기



황금잉어가물치 3

김성엽 감사합니다.

2022.11.12. 17:55 답글쓰기



카일 3

강의 잘 들었습니다^^



김성엽 M 작성자

파이팅입니다 ㅎㅎ



2022.11.19. 00:33 답글쓰기



cpkjk 3

좋은 설명 감사합니다~ 원리를 알게 돼서 신기하네요 ㅎㅎ 이어지는 강의들이 기대됩니다!!
기록용:221128_정독_2회차

2022.11.28. 20:24 답글쓰기



김성엽 M 작성자

문법의 기본 의도를 잘 파악하면 언어의 힘을 제대로 활용할수 있게 됩니다~ :)



2022.11.28. 23:03 답글쓰기



whale83 3

안녕하세요 선생님! 강의를 보다가 궁금한게 있어서요!
this포인터를 사용하기 전에는
struct VluaList

```
{
int a,b,c;
};
```

로 struct ValueList의 범위가 짧으데 왜 this 포인터를 사용한 이후에는 아래 사진처럼 struct ValueList의 범위가 늘어나는지 궁금합니다~
그리고 왜 위에 코드처럼 범위를 짧게 사용하면 안되는지도 궁금합니다!



2024.03.08. 17:32 답글쓰기



김성엽 M 작성자

강좌에서도 설명이 나오듯이 함수의 소속을 변경해서 실제로 포인터 문법이 사용되더라도 코드에서는 포인터 문법을 감추려고 이렇게 스타일을 변경한 것입니다. 즉, MultiplyValue 함수에 전달되는 struct ValueList *p_values 표현을 숨기기 위해 MultiplyValue 함수를 ValueList 내부로 옮긴다음 values.MultiplyValue(); 형태로 호출 표현을 변경해서 포인터 문법이 눈에 보이지 않도록 변경한 것입니다.

해석기 입장에서는 "MultiplyValue 함수가 values 변수의 주소를 사용"한다라고 표현한 것이나 "values 변수의 MultiplyValue 함수를 사용"한다 표현을 동일한 명령으로 번역할 수 있기 때문에 함수의 위치를 구조체 안쪽으로 옮겨서 표현을 단순하게 만든 것입니다.

MultiplyValue(&values); 라고 적은 것이나 values.MultiplyValue(); 라고 적은 것이나 해석기 입장에서는 동일하게 처리할 수 있습니다.

2024.03.09. 00:18 답글쓰기



김성엽 M 작성자

결국 C++ 언어는 표기를 좀더 쉽게 변경해서 프로그래밍을 더 쉽게 할수 있도록 만들어 놓았는데 표기를 쉽게 한다는 것은 최대한 감출수 있다면 포인터 표기를 감추는 것이고 서로 관련이 있는 것들은 최대한 한 곳에 모아서 표현해서 코드를 좀더 보기 편하게 만드는 것입니다.

2024.03.09. 00:20 답글쓰기



김성엽 M 작성자

C 언어 스타일에서 struct ValueList와 MultiplyValue 함수는 서로 떨어져서 선언되어 있지만 MultiplyValue 함수는 매개 변수로 struct ValueList를 사용하기 때문에 struct ValueList 구조체가 없으면 사용이 불가능한 함수입니다. 즉, MultiplyValue 함수는 struct ValueList에 종속된 표현이기 때문에 서로 떨어져 있어도 같이 참조하면서 프로그래밍해야하기 때문에 struct ValueList에 MultiplyValue 함수를 포함시켜서 보여주는 것이 프로그래머 입장에서는 소스를 보기가 더 편할 것입니다.

2024.03.09. 00:23 답글쓰기



김성엽 M 작성자

그리고 이 강좌는 C 언어와 C++ 언어의 프로그래밍에 대한 시각차이를 설명하기 위한 것이기 때문에 내용이 익숙하지 않다면 최소 3번 이상은 반복해서 보는 것을 추천합니다. 실제로 C++ 언어로 프로그래밍하는 분들중에 this 포인터가 왜 만들어졌는지 잘 모르시는 분들이 더 많습니다. 그러니 제가 무엇을 설명하고 싶어하는지 꼭 알아내시기 바랍니다 :)

2024.03.09. 00:25 답글쓰기



whale83 3

김성엽 님! 설명 읽고 다시 영상 보니 완전히 이해했습니다! 감사합니다~

2024.03.09. 16:12 답글쓰기



김성엽 M 작성자

whale83 파이팅입니다 :)



수고하셨습니다

2024.03.09. 20:37 답글쓰기

dh221009

댓글을 남겨보세요



등록