C++ 강좌 16회 : C++ 언어에서 달라진 static, const 키워드 사용법

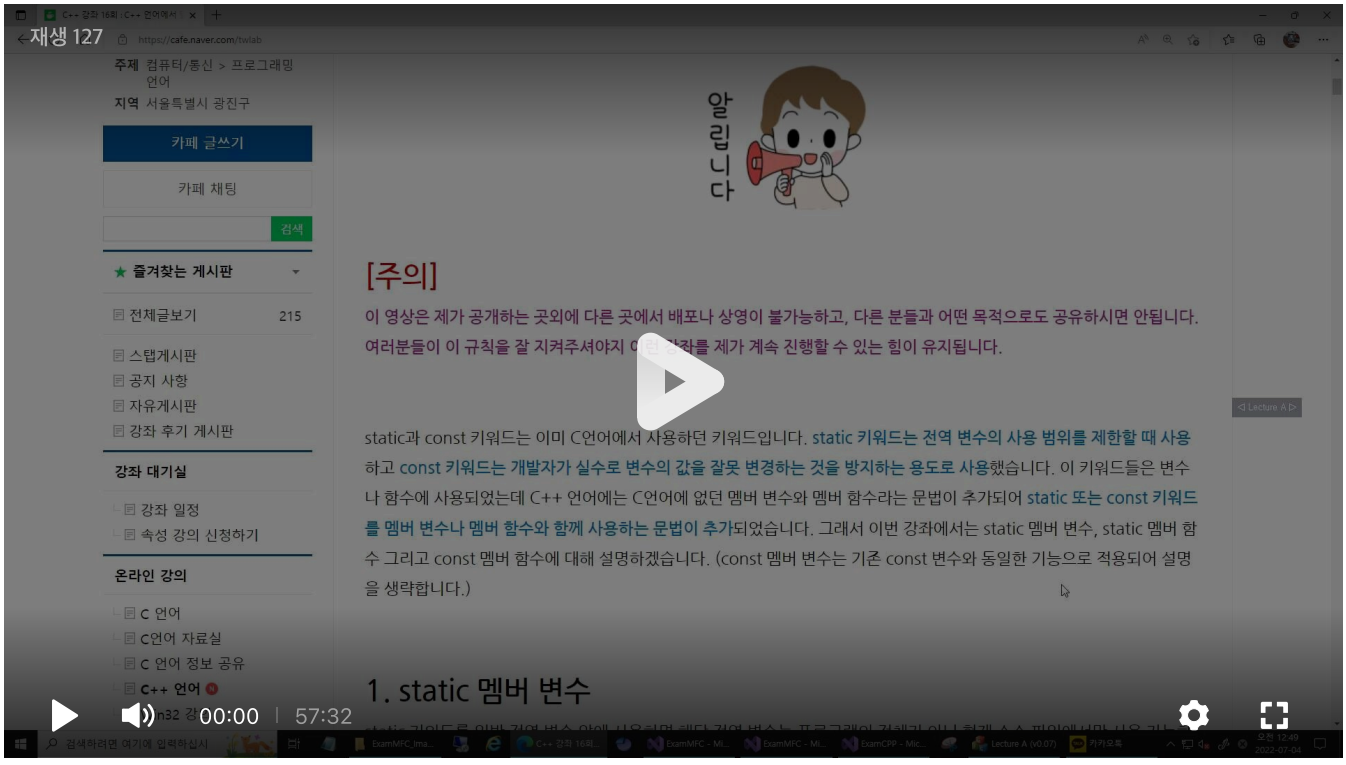
김성엽 카페매니저 + 구독 1:1 채팅

2022.07.03. 17:05 조회 364

[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.

static과 const 키워드는 이미 C언어에서 사용하던 키워드입니다. static 키워드는 전역 변수의 사용 범위를 제한할 때 사용하고 const 키워드는 개발자가 실수로 변수의 값을 잘못 변경하는 것을 방지하는 용도로 사용했습니다. 이 키워드들은 변수나 함수에 사용되었는데 C++ 언어에는 C언어에 없던 멤버 변수와 멤버 함수라는 문법이 추가되어 static 또는 const 키워드를 멤버 변수나 멤버 함수와 함께 사용하는 문법이 추가되었습니다. 그래서 이번 강좌에서는 static 멤버 변수, static 멤버 함수 그리고 const 멤버 함수에 대해 설명하겠습니다. (const 멤버 변수는 기존 const 변수와 동일한 기능으로 적용되어 설명을 생략합니다.)



C++ 강좌 16회 : static, const 키워드

1. static 멤버 변수

static 키워드를 일반 전역 변수 앞에 사용하면 해당 전역 변수는 프로그램의 전체가 아닌 현재 소스 파일에서만 사용 가능하게 사용 범위가 제한됩니다. 그리고 이렇게 static 키워드를 붙인 전역 변수를 특정 함수 내에 적게 되면 이 전역 변수는 해당 함수에서만 사용이 가능한 전역 변수가 됩니다. 만약, 이 키워드에 대해 잘 모르신다면 아래에 링크한 동영상 강좌를 보시거나 'Do it! C언어 입문'책의 11장은 참고하시기 바랍니다.

클래스 내부에 선언해서 사용하는 멤버 변수는 이 클래스로 만든 객체들간에 공유되는 개념이 아니라 각 객체에 개별적으로 존재합니다. 예를 들어, 아래와 같이 Tipsware 클래스를 만들고 m_object_count 멤버 변수를 선언했다면 Tipsware 클래스로 만든 객체들은 각각의 객체 내부에 m_object_count 변수를 위한 메모리 공간을 별도로 가지고 있습니다. 따라서 temp1 객체의 m_object_count 값은 변경한다고 해서 temp2 객체의 m_object_count 멤버 변수의 값이 변경되지 않는 다는 뜻입니다.

```
#include <stdio.h>    // printf 함수를 사용하기 위해

class Tipsware
{
private:
    int m_object_count;  // 멤버 변수

public:
    Tipsware()  // 생성자
    {
        m_object_count = 0;
    }

    // 객체 외부에서 m_object_count 변수의 값을 변경할 때 사용하는 함수
    void SetObjectCount(int a_count)
    {
        m_object_count = a_count;
    }

    // 객체 외부에서 m_object_count 변수의 값을 얻을 때 사용하는 함수
    int GetObjectCount()
    {
        return m_object_count;
    }
};

int main()
{
    // Tipsware 클래스를 사용하여 객체를 두 개 만든다.
    Tipsware temp1, temp2;

    // temp1 객체의 m_object_count 값을 5로 변경한다.
    temp1.SetObjectCount(5);

    // temp1과 temp2 객체의 m_object_count 값을 출력한다.
    printf("temp1: %d, temp2: %d\n", temp1.GetObjectCount(), temp2.GetObjectCount());
    return 0;
}
```

위 예제는 아래와 같이 실행됩니다. temp1, temp2 객체는 각각의 m_object_count 변수를 가지고 있기 때문에 temp1 객체의 m_object_count 값을 수정한다고해서 temp2 객체의 m_object_count 변수 값이 변경되지 않습니다.

```
temp1: 5, temp2: 0

이 창을 닫으려면 아무 키나 누르세요...
```

그런데 이런 특징은 객체의 독립성을 보장이라는 개념에서는 좋겠지만 **동일한 클래스로 만들어진 객체들만 공유할 수 있는 메모리 개념이 필요한 경우에는 전역 변수를 사용**해야 하기 때문에 객체 개념에 모순이 생기게 됩니다. 예를 들어, 위 예제에

서 Tipsware 클래스로 만든어진 객체의 수를 파악하고 싶은 경우, 클래스의 멤버 변수는 각자의 메모리를 가지기 때문에 객체 간에 정보를 공유할 수 없어 객체의 수를 계산할수 없습니다. 그래서 아래와 같이 전역 변수를 사용해야 합니다.

```cpp
#include <stdio.h>    // printf 함수를 사용하기 위해

// Tipsware 클래스로 만들어진 객체의 수를 파악하기 위해 사용될 전역 변수
int g_object_count = 0;

class Tipsware
{
private:
    int m_object_count;  // 멤버 변수

public:
    Tipsware()  // 생성자
    {
        m_object_count = 0;
        // g_object_count 전역 변수의 값을 1 증가
        g_object_count++;
    }

    ~Tipsware()  // 파괴자
    {
        // g_object_count 전역 변수의 값을 1 감소
        g_object_count--;
    }

    // 객체 외부에서 m_object_count 변수의 값을 변경할 때 사용하는 함수
    void SetObjectCount(int a_count)
    {
        m_object_count = a_count;
    }

    // 객체 외부에서 m_object_count 변수의 값을 얻을 때 사용하는 함수
    int GetObjectCount()
    {
        return m_object_count;
    }
};

int main()
{
    // Tipsware 클래스를 사용하여 객체를 두 개 만든다.
    Tipsware temp1, temp2;
    printf("Tipsware로 만들어진 객체의 개수: %d\n", g_object_count);

    // Tipsware 클래스를 사용하여 객체를 추가로 만든다.
    Tipsware temp3;
    printf("Tipsware로 만들어진 객체의 개수: %d\n", g_object_count);

    // Tipsware 클래스를 사용하여 객체를 동적으로 추가한다.
    Tipsware *p = new Tipsware;
    printf("Tipsware로 만들어진 객체의 개수: %d\n", g_object_count);

    // 동적으로 추가한 객체를 제거한다.
```

```
    delete p;
    printf("Tipsware로 만들어진 객체의 개수: %d\n", g_object_count);

    return 0;
}
```

위 예제는 아래와 같이 출력됩니다. temp1, temp2만 선언되었을 때는 g_object_count 값이 2로 출력되고 temp3 객체가 추가되면 g_object_count값이 3이 됩니다. 그리고 g_object_count의 값은 객체의 생성자에서 1증가하기 때문에 일반 변수가 아닌 동적 메모리 할당 기술로 객체를 추가하더라도 값이 증가하는 것은 동일합니다. 따라서 new 연산자로 객체를 추가해도 g_object_count 값은 4로 증가합니다. 그리고 파괴자에서 g_object_count 값이 1 감소하도록 되어 있기 때문에 delete 연산자로 동적할당된 객체를 제거하면 g_object_count 값은 3으로 변경됩니다.

```
Tipsware로 만들어진 객체의 개수: 2
Tipsware로 만들어진 객체의 개수: 3
Tipsware로 만들어진 객체의 개수: 4
Tipsware로 만들어진 객체의 개수: 3

이 창을 닫으려면 아무 키나 누르세요...
```

그런데 **함수가 종결될 때 사라지는 지역 변수의 특징 때문에 지역 변수로 선언된 객체의 파괴자는 함수가 끝나야지 호출**됩니다. 따라서 main 함수에 출력문을 추가하는 방법으로는 g_object_count 변숫값이 0이 되는 것을 확인할 수 없습니다. 따라서 g_object_count 변수의 값이 0이 되는 것을 확인하고 싶다면 아래와 같이 출력문을 객체의 파괴자로 옮기면 좀더 정확하게 확인할 수 있습니다. 변경하면서 생성자에도 출력문을 추가하여 main 함수에 사용된 출력문은 모두 제거했습니다.

```cpp
#include <stdio.h>    // printf 함수를 사용하기 위해

// Tipsware 클래스로 만들어진 객체의 수를 파악하기 위해 사용될 전역 변수
int g_object_count = 0;

class Tipsware
{
private:
    int m_object_count;  // 멤버 변수

public:
    Tipsware()  // 생성자
    {
        m_object_count = 0;
        // g_object_count 전역 변수의 값을 1 증가하고 해당 값을 출력
        printf("Tipsware로 만들어진 객체의 개수: %d\n", ++g_object_count);
    }

    ~Tipsware()  // 파괴자
    {
        // g_object_count 전역 변수의 값을 1 감소하고 해당 값을 출력
        printf("Tipsware로 만들어진 객체의 개수: %d\n", --g_object_count);
    }

    // 객체 외부에서 m_object_count 변수의 값을 변경할 때 사용하는 함수
    void SetObjectCount(int a_count)
    {
        m_object_count = a_count;
    }

    // 객체 외부에서 m_object_count 변수의 값을 얻을 때 사용하는 함수
    int GetObjectCount()
    {
        return m_object_count;
    }
};

int main()
{
    // Tipsware 클래스를 사용하여 객체를 두 개 만든다.
    Tipsware temp1, temp2;
    // Tipsware 클래스를 사용하여 객체를 추가로 만든다.
    Tipsware temp3;

    // Tipsware 클래스를 사용하여 객체를 동적으로 추가한다.
    Tipsware *p = new Tipsware;
    // 동적으로 추가한 객체를 제거한다.
    delete p;

    return 0;
}
```

위 예제는 아래와 같이 출력됩니다.

```
Tipsware로 만들어진 객체의 개수: 1
Tipsware로 만들어진 객체의 개수: 2
Tipsware로 만들어진 객체의 개수: 3
Tipsware로 만들어진 객체의 개수: 4
Tipsware로 만들어진 객체의 개수: 3
Tipsware로 만들어진 객체의 개수: 2
Tipsware로 만들어진 객체의 개수: 1
Tipsware로 만들어진 객체의 개수: 0

이 창을 닫으려면 아무 키나 누르세요...
```

그런데 이렇게 **객체가 객체 외부에 선언된 전역 변수를 사용하게 되면 객체의 독립성이 훼손**됩니다. 왜냐하면 g_object_count 변수는 이 객체에서만 사용 가능한 것이 아니라 다른 함수나 다른 객체에서도 사용이 가능하기 때문에 실수 또는 고의로 g_object_count 변수의 값이 변경될 수 있기 때문입니다. 따라서 **전역 변수를 사용하더라도 특정 클래스로 만들어진 객체들만 사용할 수 있는 전역 변수 표현이 추가되었는데 그 문법이 static 멤버 변수**입니다. 결국 static 키워드로 제한하는 전역 변수의 사용 범위가 '특정 소스'나 '특정 함수'뿐만 아니라 '특정 클래스'로 제한하는 기능이 추가된 것입니다. 그리고 다른 관점에서 보면 동일한 클래스로 만들어진 객체들 간에 사용할 수 있는 공유 메모리가 추가된 것이기도 합니다.

Tipsware 클래스로 만들어진 객체의 수를 계산하는 예제를 static 멤버 변수를 사용해서 재구성해보면 다음과 같습니다. m_object_count 변수는 외부에서 조작하면 안되기 때문에 SetObjectCount , GetObjectCount 함수는 제거했습니다. 그리고 **static 멤버 변수는 Tipsware 클래스로 만들어진 객체들 간에 공유되기 때문에 특정 클래스가 소유하고 있지 않아 초깃값은 클래스 내부에서 지정할 수 없습니다.** 그래서 Tipsware:: 네임스페이스를 사용해 클래스 외부에 적습니다.

```cpp
#include <stdio.h>    // printf 함수를 사용하기 위해

class Tipsware
{
private:
    static int m_object_count;  // static 멤버 변수

public:
    Tipsware()  // 생성자
    {
        // m_object_count 변수의 값을 1 증가하고 해당 값을 출력
        printf("Tipsware로 만들어진 객체의 개수: %d\n", ++m_object_count);
    }

    ~Tipsware()  // 파괴자
    {
        // m_object_count 변수의 값을 1 감소하고 해당 값을 출력
        printf("Tipsware로 만들어진 객체의 개수: %d\n", --m_object_count);
    }
};

// Tipsware 클래스로 만들어진 객체의 수를 파악하기 위해 사용될
// static 멤버 변수는 객체들간에 공유되기 때문에 초기화를 특정 객체에서
// 할 수 없어 객체 외부에서 초기화한다.
int Tipsware::m_object_count = 0;

int main()
{
    // Tipsware 클래스를 사용하여 객체를 두 개 만든다.
    Tipsware temp1, temp2;
    // Tipsware 클래스를 사용하여 객체를 추가로 만든다.
    Tipsware temp3;

    // Tipsware 클래스를 사용하여 객체를 동적으로 추가한다.
    Tipsware *p = new Tipsware;
    // 동적으로 추가한 객체를 제거한다.
    delete p;

    return 0;
}
```

위 예제는 아래와 같이 출력됩니다.

```
Tipsware로 만들어진 객체의 개수: 1
Tipsware로 만들어진 객체의 개수: 2
Tipsware로 만들어진 객체의 개수: 3
Tipsware로 만들어진 객체의 개수: 4
Tipsware로 만들어진 객체의 개수: 3
Tipsware로 만들어진 객체의 개수: 2
Tipsware로 만들어진 객체의 개수: 1
Tipsware로 만들어진 객체의 개수: 0

이 창을 닫으려면 아무 키나 누르세요...
```

이와 같이 **특정 클래스로 만든어진 객체든간에 정보른 공유하고 싶다면 static 멤버 변수른 사용**하면 됩니다. 그리고 이렇게 선언된 static 멤버 변수는 객체 표현을 훼손하지 않으면서도 전역 변수를 사용할 수 있게 해주는 문법입니다.

2. static 멤버 함수

지금까지는 클래스의 멤버 함수를 호출하면 숨겨진 매개 변수인 this 포인터가 사용된다고 설명했습니다. 그런데 프로그램을 하다보면 **클래스의 멤버 함수에서 멤버 변수른 사용하지 않는 경우도 있은덴데 이런 경우에는 의미없이 this 포인터가 매개 변수로 넘어간 것**입니다. 즉, 클래스의 멤버 함수에 반드시 this 포인터가 선언되어 사용되면 손해인 경우가 있을 수도 있습니다. 그래서 멤버 함수에서 this 포인터를 사용하고 싶지 않다면 **멤버 함수 앞에 static 키워드른 추가해 멤버 함수에서 this 포인터의 사용은 제한**할 수 있습니다.

예를 들어, 아래와 같이 MyData 클래스에 Swap 멤버 함수를 추가했고 Swap 함수를 호출하면 m_data1, m_data2 변수의 값이 서로 바뀝니다.

```cpp
#include <stdio.h>    // printf 함수를 사용하기 위해

class MyData
{
private:
    int m_data1, m_data2; // 멤버 변수

public:
    MyData()  // 생성자
    {
        m_data1 = m_data2 = 0;
    }

    // m_data1, m_data2 변수의 값을 서로 바꾸는 함수
    void Swap()
    {
        int temp = m_data1;
        m_data1 = m_data2;
        m_data2 = temp;
    }

    // m_data1, m_data2 변수에 값을 대입하는 함수
    void SetData(int a_data1, int a_data2)
    {
        m_data1 = a_data1;
        m_data2 = a_data2;
    }

    // m_data1 변수의 값을 얻는 함수
    int GetData1()
    {
        return m_data1;
    }

    // m_data2 변수의 값을 얻는 함수
    int GetData2()
    {
        return m_data2;
    }
};

int main()
{
    MyData temp;
    // temp 객체의 두 멤버 변수의 값을 출력한다.
    printf("m_data1: %d, m_data2: %d\n", temp.GetData1(), temp.GetData2());

    // 두 멤버 변수에 2, 5 값을 대입한다.
    temp.SetData(2, 5);
    // temp 객체의 두 멤버 변수의 값을 출력한다.
    printf("m_data1: %d, m_data2: %d\n", temp.GetData1(), temp.GetData2());
```

```
    // 두 멤버 변수의 값을 서로 바꾼다.
    temp.Swap();
    // temp 객체의 두 멤버 변수의 값을 출력한다.
    printf("m_data1: %d, m_data2: %d\n", temp.GetData1(), temp.GetData2());

    return 0;
}
```

위 예제는 아래와 같이 출력됩니다.

```
m_data1: 0, m_data2: 0
m_data1: 2, m_data2: 5
m_data1: 5, m_data2: 2

이 창을 닫으려면 아무 키나 누르세요...
```

그런데 위 코드에서 Swap 함수가 클래스의 특정 멤버 변수가 아닌 범용적으로 사용되어야 한다면 아래와 같이 코드를 수정해야 합니다.

```
#include <stdio.h>    // printf 함수를 사용하기 위해

class MyData
{
private:
    int m_data1, m_data2; // 멤버 변수
    int m_value1, m_value2; // 멤버 변수

public:
    MyData()  // 생성자
    {
        m_data1 = m_data2 = 0;
        m_value1 = m_value2 = 0;
    }

    // 두 포인터 가리키는 대상의 값을 변경하는 함수
    void Swap(int *ap_a, int *ap_b)
    {
        int temp = *ap_a;
        *ap_a = *ap_b;
        *ap_b = temp;
    }

    // m_data1, m_data2 변수의 값을 서로 바꾸는 함수
    void DataSwap()
    {
        Swap(&m_data1, &m_data2);
    }

    // m_value1, m_value2 변수의 값을 서로 바꾸는 함수
    void ValueSwap()
    {
        Swap(&m_value1, &m_value2);
    }

    // m_data1, m_data2 변수에 값을 대입하는 함수
    void SetData(int a_data1, int a_data2)
    {
        m_data1 = a_data1;
        m_data2 = a_data2;
    }

    // m_value1, m_value2 변수에 값을 대입하는 함수
    void SetValue(int a_value1, int a_value2)
    {
        m_value1 = a_value1;
        m_value2 = a_value2;
    }

    // m_data1, m_data2 변수의 값을 출력하는 함수
    void PrintData()
    {
```

```
        printf("m_data1: %d, m_data2: %d\n", m_data1, m_data2);
    }

    // m_value1, m_value2 변수의 값을 출력하는 함수
    void PrintValue()
    {
        printf("m_value1: %d, m_value2: %d\n", m_value1, m_value2);
    }
};

int main()
{
    MyData temp;
    // m_data1, m_data2 멤버 변수에 2, 5 값을 대입한다.
    temp.SetData(2, 5);
    // temp 객체의 m_data1, m_data2 멤버 변숫값을 출력한다.
    temp.PrintData();
    // m_data1, m_data2 멤버 변수의 값을 서로 바꾼다.
    temp.DataSwap();
    // temp 객체의 m_data1, m_data2 멤버 변숫값을 출력한다.
    temp.PrintData();

    // m_value1, m_value2 멤버 변수에 3, 1 값을 대입한다.
    temp.SetValue(3, 1);
    // temp 객체의 m_value1, m_value2 멤버 변숫값을 출력한다.
    temp.PrintValue();
    // m_value1, m_value2 멤버 변수의 값을 서로 바꾼다.
    temp.ValueSwap();
    // temp 객체의 m_value1, m_value2 멤버 변숫값을 출력한다.
    temp.PrintValue();

    return 0;
}
```

위 예제는 아래와 같이 출력됩니다.

```
m_data1: 2, m_data2: 5
m_data1: 5, m_data2: 2
m_value1: 3, m_value2: 1
m_value1: 1, m_value2: 3

이 창을 닫으려면 아무 키나 누르세요...
```

그런데 이렇게 변경된 Swap 함수는 **두 가지 단점**을 가지고 있습니다. 첫 번째 단점은 Swap 함수는 MyData 클래스의 멤버 함수이기 때문에 **Swap 함수를 사용하려면 MyData 클래스로 선언된 객체가 있어야 한다**는 것입니다. 예를 들어, MyData 클래스에 추가된 Swap 함수는 변수의 주소만 넘기면 사용 가능하기 때문에 아래와 같이 지역 변수의 주소를 넘겨서 사용하는 것도 가능합니다. 하지만 **단순히 Swap 함수를 사용하려고 temp 객체를 선언하게 되면 사용되지 않는 메모리가 할당되고 의미없이 객체의 생성자와 파괴자가 호출**됩니다.

```
#include <stdio.h>    // printf 함수를 사용하기 위해

class MyData
{
    // ... 내부 코드 생략 ...
};

int main()
{
    int a = 5, b = 7;
    MyData temp;  // Swap 함수를 사용하기 위해 선언

    // a, b 변수의 값을 출력한다.
    printf("a: %d, b: %d\n", a, b);
    // a, b의 값을 서로 바꾼다.
    temp.Swap(&a, &b);
    // a, b 변수의 값을 출력한다.
    printf("a: %d, b: %d\n", a, b);

    return 0;
}
```

위 예제는 아래와 같이 출력됩니다.

```
a: 5, b: 7
a: 7, b: 5

이 창을 닫으려면 아무 키나 누르세요...
```

그리고 **두 번째 단점은 MyData 클래스의 Swap 함수에는 멤버 변수가 사용되지 않았기 때문에 내부적으로 this 포인터가 사용되지 않습니다.** 하지만 C++ 언어의 문법은 Swap 함수가 MyData 클래스의 멤버 함수이기 때문에 this 포인터가 내부적으로 선언되어 불필요한 메모리가 사용됩니다. 그리고 이 메모리(this 포인터)에 주소가 복사되는 코드도 추가되기 때문에 불필요한 코드도 추가되는 것입니다.

따라서 **C++ 언어는 이 두 가지 단점은 해결하기 위해 static 멤버 함수라는 문법은 제공**합니다. 예를 들어, 아래와 같이 MyData 클래스의 **Swap 함수에 static은 붙여주면 Swap 함수는 이제 static 멤버 함수**가 되며 내부적으로는 this 포인터가 추가되지 않기 때문에 두 번째로 이야기한 단점이 제거됩니다. 그리고 Swap 함수에서는 멤버 변수를 사용하지 않기 때문에 예제 프로그램은 정상적으로 잘 컴파일되고 실행됩니다.

```
#include <stdio.h>    // printf 함수를 사용하기 위해

class MyData
{
    // ... 내부 코드 생략 ...

    // 두 포인터 가리키는 대상의 값을 변경하는 static 멤버 함수
    // 이 함수에서는 this 포인터가 제공되지 않는다.
    static void Swap(int *ap_a, int *ap_b)
    {
        int temp = *ap_a;
        *ap_a = *ap_b;
        *ap_b = temp;
    }

    // ... 내부 코드 생략 ...
};

int main()
{
    int a = 5, b = 7;
    MyData temp;  // Swap 함수를 사용하기 위해 선언

    // a, b 변수의 값을 출력한다.
    printf("a: %d, b: %d\n", a, b);
    // a, b의 값을 서로 바꾼다.
    temp.Swap(&a, &b);
    // a, b 변수의 값을 출력한다.
    printf("a: %d, b: %d\n", a, b);

    return 0;
}
```

그리고 Swap 함수는 이제 static 멤버 함수이기 때문에 this 포인터를 사용되지 않아 이 함수를 사용할 때 객체를 생성할 필요가 없습니다. 즉, 객체의 주소를 Swap 함수에 넘길 필요가 없기 때문에 MyData 객체를 생성할 필요가 없다는 뜻입니다. 하지만 Swap 함수의 소속이 MyData 클래스이기 때문에 아래와 같이 영역 연산자(스코프 연산자)를 사용해서 소속만 표시해주면 됩니다. 따라서 이런 표현으로 첫 번째 단점도 제거됩니다.

```
#include <stdio.h>    // printf 함수를 사용하기 위해

class MyData
{
    // ... 내부 코드 생략 ...

    // 두 포인터 가리키는 대상의 값을 변경하는 static 멤버 함수
    // 이 함수에서는 this 포인터가 제공되지 않는다.
    static void Swap(int *ap_a, int *ap_b)
    {
        int temp = *ap_a;
        *ap_a = *ap_b;
        *ap_b = temp;
    }

    // ... 내부 코드 생략 ...
};

int main()
{
    int a = 5, b = 7;

    // a, b 변수의 값을 출력한다.
    printf("a: %d, b: %d\n", a, b);
    // a, b의 값을 서로 바꾼다.
    MyData::Swap(&a, &b);  // this 포인터를 사용하지 않기 때문에 이런 표현이 가능하다.
    // a, b 변수의 값을 출력한다.
    printf("a: %d, b: %d\n", a, b);

    return 0;
}
```

3. static 멤버 변수와 static 멤버 함수

static 멤버 변수는 특정 객체에 포함된 것이 아니라, 객체들이 공유하는 전역 변수일 뿐입니다. 따라서 this로 접근하는 개념이 아닌 클래스의 네임스페이스로 접근하는 개념입니다. 그래서 static 멤버 변수만 사용하는 함수에서는 this 포인터가 사용되지 않기 때문에 이런 함수는 static 멤버 함수로 선언하는 것이 좋습니다.

예를 들어, 1번에서 소개한 Tipsware 클래스에서 m_object_count 멤버 변수는 private 접근 지정자를 사용해서 선언했기 때문에 이 변수의 값을 외부에서 사용하려면 아래와 같이 GetObjectCount 같은 멤버 함수를 추가해야 합니다.

```cpp
#include <stdio.h>    // printf 함수를 사용하기 위해

class Tipsware
{
private:
    static int m_object_count;  // static 멤버 변수

public:
    Tipsware()  // 생성자
    {
        // m_object_count 변수의 값을 1 증가하고 해당 값을 출력
        printf("Tipsware로 만들어진 객체의 개수: %d\n", ++m_object_count);
    }

    ~Tipsware()  // 파괴자
    {
        // m_object_count 변수의 값을 1 감소하고 해당 값을 출력
        printf("Tipsware로 만들어진 객체의 개수: %d\n", --m_object_count);
    }

    // static 멤버 변수의 값을 객체 외부에 제공할 때 사용하는 함수
    int GetObjectCount()
    {
        return m_object_count;
    }
};

// Tipsware 클래스로 만들어진 객체의 수를 파악하기 위해 사용될
// static 멤버 변수는 객체들간에 공유되기 때문에 초기화를 특정 객체에서
// 할 수 없어 객체 외부에서 초기화한다.
int Tipsware::m_object_count = 0;

int main()
{
    // GetObjectCount 함수를 호출하기 위해 객체를 선언한다.
    Tipsware temp;

    printf("m_object_count : %d\n", temp.GetObjectCount());

    return 0;
}
```

위 예제는 아래와 같이 출력됩니다.

```
Tipsware로 만들어진 객체의 개수: 1
m_object_count : 1
Tipsware로 만들어진 객체의 개수: 0

이 창을 닫으려면 아무 키나 누르세요...
```

하지만 GetObjectCount 함수를 일반 멤버 함수로 추가하면 이 함수를 사용하기 위해서는 Tipsware 객체가 반드시 필요해서 위 예제처럼 temp 객체를 선언해야지만 사용이 가능합니다. 따라서 m_object_count 값이 0인 경우는 확인할 수 없습니다. 하지만 아래와 같이 GetObjectCount 함수를 static 멤버 함수로 선언하면 this 포인터를 사용하지 않기 때문에 이 함수를 호출하기 위해 Tipsware 클래스로 만든 객체가 필요 없어집니다. 즉, Tipsware::처럼 네임스페이스만 추가해서 GetObjectCount 함수를 사용할 수 있기 때문에 m_object_count 값이 0인 경우도 얻을 수 있습니다.

```cpp
#include <stdio.h>    // printf 함수를 사용하기 위해

class Tipsware
{
private:
    static int m_object_count;  // static 멤버 변수

public:
    Tipsware()  // 생성자
    {
        // m_object_count 변수의 값을 1 증가하고 해당 값을 출력
        printf("Tipsware로 만들어진 객체의 개수: %d\n", ++m_object_count);
    }

    ~Tipsware()  // 파괴자
    {
        // m_object_count 변수의 값을 1 감소하고 해당 값을 출력
        printf("Tipsware로 만들어진 객체의 개수: %d\n", --m_object_count);
    }

    // static 멤버 변수의 값을 객체 외부에 제공할 때 사용하는 함수
    // 이 함수는 static 함수이기 때문에 내부적으로 this 포인터가 제공되지 않습니다.
    static int GetObjectCount()
    {
        return m_object_count;
    }
};

// Tipsware 클래스로 만들어진 객체의 수를 파악하기 위해 사용될
// static 멤버 변수는 객체들간에 공유되기 때문에 초기화를 특정 객체에서
// 할 수 없어 객체 외부에서 초기화한다.
int Tipsware::m_object_count = 0;

int main()
{
    // GetObjectCount 함수는 static 멤버 함수라서 객체가 없어도 호출 가능
    printf("m_object_count : %d\n", Tipsware::GetObjectCount());

    // temp 객체를 선언한다.
    Tipsware temp;
    // temp.GetObjectCount()도 가능하고 Tipsware::GetObjectCount()도 가능
    printf("m_object_count : %d\n", temp.GetObjectCount());

    return 0;
}
```

위 예제는 아래와 같이 출력됩니다.

```
m_object_count : 0
Tipsware로 만들어진 객체의 개수: 1
m_object_count : 1
Tipsware로 만들어진 객체의 개수: 0

이 창을 닫으려면 아무 키나 누르세요...
```

결론적으로 static 멤버 변수를 사용하는 함수는 static 멤버 함수로 선언하면 활용도가 더 높아집니다. 하지만 static 멤버 함수는 this 포인터 사용이 불가능하기 때문에 이 함수에서는 일반 멤버 변수를 사용하면 오류가 발생합니다.

4. const 멤버 함수

아래와 같이 한 개의 정숫값을 관리하는 Tipssoft 클래스가 있다고 가정하겠습니다.

```
#include <stdio.h>    // printf 함수를 사용하기 위해

class Tipssoft
{
private:
    int m_data;

public:
    Tipssoft()  // 생성자
    {
        m_data = 0;
    }

    // 외부에서 m_data 변수에 값을 대입할 때 사용할 함수
    void SetData(int a_data)  // Tipssoft * const this;
    {
        m_data = a_data;  // this->m_data = a_data;
    }

    // 외부에서 m_data 변수의 값을 얻을 때 사용할 함수
    int GetData() // Tipssoft * const this;
    {
        return m_data;  // return this->m_data;
    }
};

int main()
{
    // temp 객체를 선언한다.
    Tipssoft temp;

    // m_data 변수에 7을 대입한다.
    temp.SetData(7);
    // m_data 변수의 값을 출력한다.
    printf("m_data: %d\n", temp.GetData());
    return 0;
}
```

위 예제는 아래와 같이 출력됩니다.

```
m_data: 7

이 창을 닫으려면 아무 키나 누르세요...
```

그런데 위 예제에서 SetData 함수는 멤버 변수의 값을 바꾸는 기능이지만 GetData 함수는 멤버 변수의 값은 수정되지 않고 단순히 읽기만 수행하기 때문에 멤버 변수의 값이 변경되지 않는다는 것은 강조하기 위해 아래와 같이 const 붙여 'const 멤버 함수'로 선언하는 경우도 있습니다. 이것은 개발자가 다른 개발자들에게 자신의 의도를 강조하기 위한 목적일뿐 꼭 이렇게 사용할 필요는 없습니다.

```
#include <stdio.h>    // printf 함수를 사용하기 위해

class Tipssoft
{
private:
    int m_data;

public:
    Tipssoft()  // 생성자
    {
        m_data = 0;
    }

    // 외부에서 m_data 변수에 값을 대입할 때 사용할 함수
    void SetData(int a_data)  // Tipssoft * const this;
    {
        m_data = a_data;  // this->m_data = a_data;
    }

    // 외부에서 m_data 변수의 값을 얻을 때 사용할 함수
    // 이 함수는 const 키워드를 사용했기 때문에 const 멤버 함수로 처리됩니다.
    // const 멤버 함수는 this 포인터에 영향을 주어 this 포인터의 자료형이 Tipssoft * const에서
    // const Tipssoft * const로 변경됩니다.
    int GetData() const // const Tipssoft * const this;
    {
        return m_data;  // return this->m_data;
    }
};

int main()
{
    // temp 객체를 선언한다.
    Tipssoft temp;

    // m_data 변수에 7을 대입한다.
    temp.SetData(7);
    // m_data 변수의 값을 출력한다.
    printf("m_data: %d\n", temp.GetData());
    return 0;
}
```

그런데 const 함수에서 const 자료형이 붙은 위치가 좀 특이합니다. C언어에서는 const int GetData()와 같은 표현은 사용해도 int GetData() const 같은 표현은 제공되지 않습니다. 여기서 const int에 사용된 const는 반환되는 값의 자료형이 const int라는 뜻입니다. 하지만 C++ 언어에서 추가로 제공되는 int GetData() const 표현에서 const는 해당 함수에 사용되는 this 포인터의 자료형에 영향을 미칩니다. 즉, 보통 멤버 함수는 해당 클래스의 포인터로 사용되는 this 포인터가 제공되기 때문에 아래와 같은 자료형을 가집니다. this 포인터에 저장된 주소는 변경할 수 없지만 this 포인터가 가리키는 대상의 값은 변경이 가능하다는 뜻입니다.

```
Tipssoft * const this;
```

하지만 const 멤버 함수에 사용되는 this 포인터는 아래와 같은 자료형을 가집니다. this 포인터에 저장된 주소도 변경할 수 없고 this 포인터가 가리키는 대상의 값도 변경할 수 없다는 뜻입니다. 즉, 멤버 변수의 값을 읽기 용도로만 사용이 가능하다는 뜻입니다.

```
const Tipssoft * const this;
```

따라서 아래와 같이 SetData 함수를 const 함수로 선언하면 수정할 수 없다고 오류가 발생합니다.

```
#include <stdio.h>    // printf 함수를 사용하기 위해

class Tipssoft
{
    // ... 코드 생략 ...

    // 외부에서 m_data 변수에 값을 대입할 때 사용할 함수
    void SetData(int a_data) const  // const Tipssoft * const this;
    {
        m_data = a_data;  // this->m_data = a_data;
    }

    // ... 코드 생략 ...;

int main()
{
    // temp 객체를 선언한다.
    Tipssoft temp;

    // m_data 변수에 7을 대입한다.
    temp.SetData(7);
    // m_data 변수의 값을 출력한다.
    printf("m_data: %d\n", temp.GetData());
    return 0;
}
```

```
빌드 시작...
1>------ 빌드 시작: 프로젝트: ExamCPP, 구성: Debug Win32 ------
1>ExamCPP.cpp
1>ExamCPP.cpp(17,9): error C3490: 'm_data'은(는) const 개체를 통해 액세스되고 있으므로 수정할 수 없습니다.
1>"ExamCPP.vcxproj" 프로젝트를 빌드했습니다. - 실패
========== 빌드: 0 성공, 1 실패, 0 최신 업데이트, 0 건너뛰기 ==========
```

5. static 멤버 함수와 const 멤버 함수

static 멤버 함수는 멤버 함수에 기본적으로 사용되는 this 포인터를 사용하지 못하게 만드는 함수이고 const 멤버 함수는 멤버 함수에 사용되는 this 포인터에 const 키워드를 추가해주는 함수입니다. 즉, static 멤버 함수는 this를 사용하지 않는 개념이고 const 멤버 함수는 this 포인터가 필요한 개념입니다. 따라서 아래와 같이 static 멤버 함수는 const 멤버로 선언할 수 없습니다. 그리고 GetData 함수는 static 멤버 함수이기 때문에 this 포인터를 사용할 수 없어서 m_data 변수를 사용하는 코드도 오류 처리됩니다. 즉, GetData 함수에서는 m_data 멤버 변수를 사용할 수 없습니다.

```
class Tipssoft
{
private:
    int m_data;

public:
    static int GetData() const // const Tipssoft * const this;
    {
        return m_data;  // return this->m_data;
    }
};
```

```
빌드 시작...
1>------ 빌드 시작: 프로젝트: ExamCPP, 구성: Debug Win32 ------
1>ExamCPP.cpp
1>ExamCPP.cpp(25,5): error C2272: 'GetData': 정적 멤버 함수에 한정자를 사용할 수 없습니다.
1>ExamCPP.cpp(26,16): error C2597: 비정적 멤버 'Tipssoft::m_data'에 대한 참조가 잘못되었습니다.
1>ExamCPP.cpp(6): message : 'Tipssoft::m_data' 선언을 참조하십시오.
1>"ExamCPP.vcxproj" 프로젝트를 빌드했습니다. - 실패
========== 빌드: 0 성공, 1 실패, 0 최신 업데이트, 0 건너뛰기 ==========
```


bac 2

클래스를 직접 만들어서 실험하던 도중 궁금한 점이 몇 가지 있어서 질문 드리려고 합니다

1. static 멤버 함수는 virtual 키워드를 쓸 수 없기 때문에 다형성을 쓸 수 없고 다형성을 쓸 수 없으니 static 멤버 함수는 오버라이딩을 쓸 수 없다고 생각하는데 static 멤버 함수는 오버라이딩을 쓸 수 없는 게 맞나요? 아니면 오버라이딩을 쓸 수 있는 다른 방법이 있나요?

2. static 멤버 함수는 오버라이딩이 불가능한 게 맞다면 static 멤버 함수는 클래스에서만 쓸 수 있는 전역 함수일 뿐 클래스에 속한 함수가 아니라서 그렇다고 생각하는데 맞나요? 만약 아니라면 static 멤버 함수가 오버라이딩이 불가능한 이유가 무엇인가요?

3. 자식 클래스에 부모 클래스의 const 멤버 함수와 반환 자료형, 함수 이름, 매개 변수가 같은 함수를 만들어도 const 멤버 함수로 만들지 않으면 부모 클래스의 const 멤버 함수를 오버라이딩한 것으로 판정되지 않는데(부모 클래스의 const 멤버 함수를 오버라이딩하려면 자식 클래스의 함수도 const 멤버 함수여야 하는 것 같은데) 눈에 보이지 않는 this 포인터 매개 변수의 자료형이 일반 멤버 함수와 달라서 반환 자료형, 함수 이름, 매개 변수가 같은 일반 멤버 함수와 const 멤버 함수가 다른 함수로 취급되는 건가요?

2022.07.14. 15:16 답글쓰기

김성엽 작성자

맞습니다. static 함수는 그냥 전역 함수인데 클래스 내에서만 사용하도록 제한된것이고 this 포인터를 활용할 수 없어서 다형성 구조에 참여할수 없습니다. 물로 다형성을 지원하는 클래스에 멤버 함수로 선언하는 것은 가능하지만 직접적인 기능구현에는 한계가 있습니다. 그래서 보통 제일 부모 클래스에서 클래스를 래핑하는 기능을 제공하는 함수를 만들거나 절대 변경되지 않는 정보를 제공하는 함수로 사용됩니다.

2022.07.14. 11:49 답글쓰기

김성엽 작성자

static 함수에는 virtual 키워드를 사용하지 못합니다. 그래서 사실상 다형성 구성에 적극적으로 참여할 수 없습니다.

2022.07.14. 11:50 답글쓰기

김성엽 작성자

사실 매개 변수에 사용된 const는 오버라이딩에 영향을 주지 않습니다. 그래서 const 가 있든 없든 동일함수로 오버라이딩이 되는데 const 멤버 함수는 예외로 적용됩니다. 사실 굳이 예외로 안해도 되었을듯한데, this 포인터를 사용한 예외처리를 더 강하게 어필하기 위해서 의도적으로 문법을 저렇게 만들지 않았나 생각해봅니다.

2022.07.14. 11:55 답글쓰기

bac 2

김성엽 1. static 멤버 함수는 virtual 키워드와 this 포인터를 쓸 수 없어서 다형성 사용이 불가능하다

2. static 멤버 함수는 다형성을 쓸 수 없으므로 자식 클래스에서 부모 클래스의 static 멤버 함수를 오버라이딩할 수 없다

3. 일반적으로 반환 자료형과 함수 이름이 같은 두 함수의 매개 변수가 개수와 형식은 같지만 한쪽에 const가 붙어 있으면 두 함수는 동일한 함수로 취급된다(ex) void Func(int a, int b), void Func(const int a, int b))

4. const 멤버 함수와 반환 자료형, 함수 이름, 매개 변수가 같은 함수는 const 멤버 함수와 this 포인터의 자료형만 다르지만 const 멤버 함수의 this 포인터가 일반적인 this 포인터와 다르다는 것을 강조하기 위해 C++은 두 함수를 다른 함수로 취급한다

5. 부모 클래스의 const 멤버 함수를 오버라이딩하려는 자식 클래스의 멤버 함수도 const 멤버 함수여야 한다

정도로 정리하면 될까요?

2022.07.14. 15:16 답글쓰기

김성엽 작성자

bac 네~ 맞습니다 ㅎ

2022.07.14. 15:18 답글쓰기

bac 2

김성엽 감사합니다

2022.07.14. 15:19 답글쓰기

a12345 2

static 멤버변수 == 객체들이 공유하는 전역변수
static 멤버함수 == this 포인터를 없애버린 멤버들이 쓰는 전역함수(네임스페이스만 있다면 객체선언없이도 사용가능)
(클래스이름)* const this == this에 들어있는 주소값 (호출한 객체의주소)를 바꾸지 마시오
const (클래스이름)* this == this가 가리키는 대상의 값을 건들지 마시오
이렇게 이해 했는데 맞을까요?

그리고 static 멤버변수도 사실상 전역변수같은거라서 해당 객체말고도 다른애들이 막 바꿀수있지만,
private 에 선언함으로써 그 객체만 쓸수있게 만드는것인가요?

private에 선언해도 프렌드함수가있으면 걔가 막 바꿀수있으니 객체 외에도 접근할 방법이 존재하는것이구요

항상 감사드립니다

2022.07.25. 16:09 답글쓰기

김성엽 M 작성자

네~ 맞습니다. 결국 프랜드 키워드 사용하기 시작하면 객체 개념 다 무너지기 때문에 제가 프랜드 언급을 안하는 겁니다 ㅎㅎ 그리고 static 멤버 변수도 전역 변수라서 초보자들에게 사용을 추천하지 않습니다. 전역 변수 자체가 나쁜 개념은 아니지만 개발자의 스타일을 좋게 형성하는데 방해가 되기 때문에 실력이 좀 좋아질때까지는 최대한 사용을 하지 않는게 좋습니다.

2022.08.15. 11:10 답글쓰기

a12345 2

김성엽 감사합니다
선생님 덕분에 개념적인 부분말고도 코드 스타일도 많이 배워갑니다!!

2022.08.15. 11:18 답글쓰기

카일 3

강의 잘 들었습니다.^^

2022.10.20. 16:38 답글쓰기

김성엽 M 작성자

수고하셨습니다

2022.10.20. 17:40 답글쓰기

임레이 3

강의를 듣고 궁금한 점이 생겨서 질문 드립니다.

1. static 멤버 변수는 Tipsware 클래스로 만들어진 객체들 간에 공유되기 때문에 특정 클래스가 소유하고 있지 않아 초깃값을 클래스 내부에서 지정할 수 없다 -> 위에 쓰인 문장 중에, 특정 클래스가 소유하고 있지 않다는 의미를 특정 객체가 소유하고 있지 않다고 이해해도 되나요?

2. static 멤버 변수 m_object_count를 초기화할 때, (int Tipsware::m_object_count = 0;) int를 빼고 그냥 Tipsware::m_object_count = 0; 이라고 하면 안되는 이유가 무엇인가요?

2023.03.21. 18:06 답글쓰기

김성엽 M 작성자

static 변수는 결국 전역변수라고 봐야합니다. 따라서 특정 객체가 소유하고 있지 않습니다. 결국 네임스페이스 개념처럼 어디에 소속되어 있다는 표현으로 사용 범위만 제한하는 걸로 보셔야 합니다.

2023.03.22. 11:15 답글쓰기

김성엽 M 작성자

따라서 초기화를 특정 객체에서 할수 있는 것이 아니기 때문에 클래스 외부에서 변수 선언형식을 사용해서 실체를 선언(전역변수 선언 형태)하고 사용합니다. 따라서 int 빼면 그냥 대입 명령문이 함수 외부에 사용되었기 때문에 오류가 됩니다. 초기화시에 사용되는 = 과 대입 연산자는 모양은 비슷하지만 사용법이 다릅니다.

지금 오류가 난건 우리가 흔히 함수 밖에 전역 변수 선언문을 제외한 다른 명령문을 사용했을때 나는 오류와 동일하다고 보시면 됩니다.

2023.03.22. 11:17 답글쓰기

임레이 3

김성엽 감사합니다!

2023.03.22. 15:26 답글쓰기

dh221009

댓글을 남겨보세요

등록