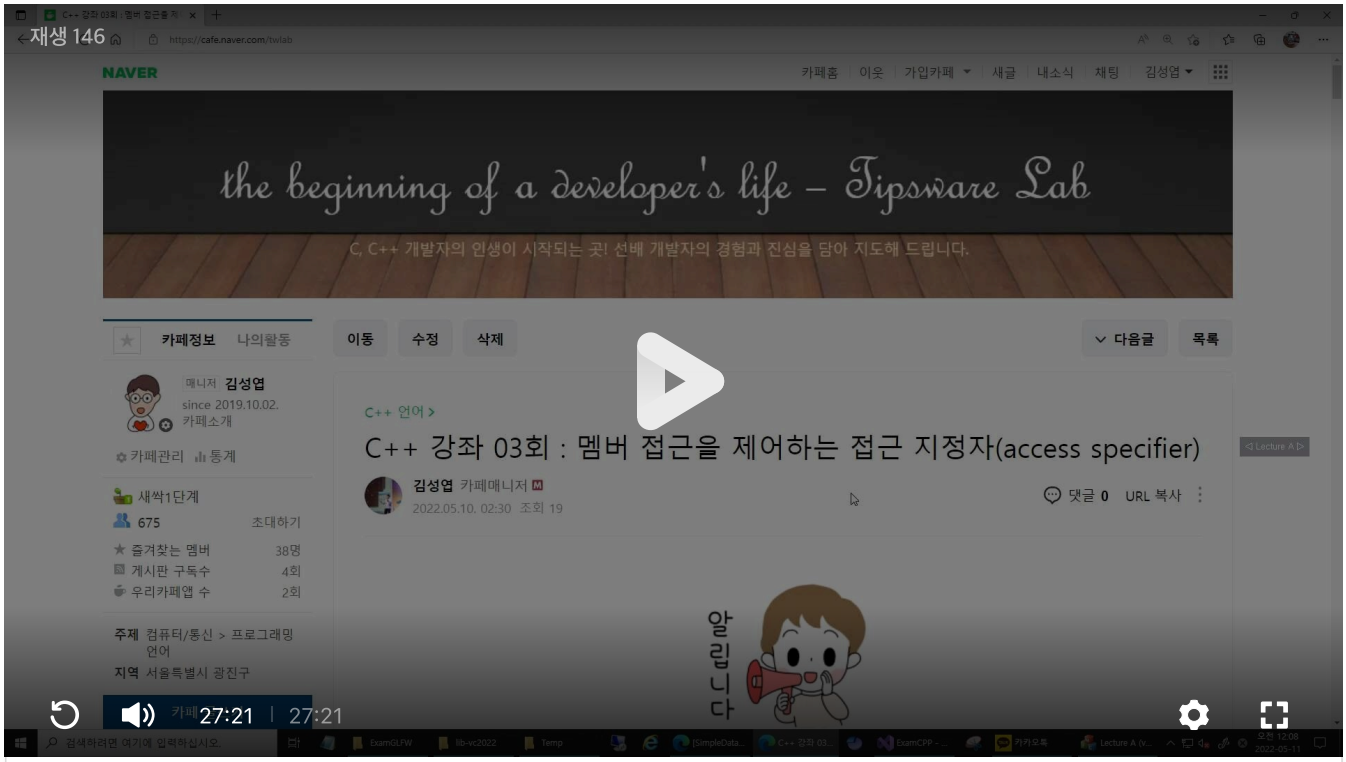
C++ 강좌 03회 : 멤버 접근을 제어하는 접근 지정자(access specifier)

김성엽 카페매니저 + 구독 1:1 채팅
2022.05.10. 02:30 조회 339

[주의]

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

이번 강좌에서는 구조체 외부에서 '멤버 변수' 또는 '멤버 함수'에 접근을 제어할 수 있도록 제공되는 '접근 지정자(access specifier)'에 대해 소개하겠습니다. C++ 언어에서는 접근 지정자로 private, protected 그리고 public 키워드를 사용할 수 있지만 아직 '상속(Inheritance)' 문법을 배우지 않았기 때문에 protected 키워드는 나중에 소개하도록 하겠습니다.



C++ 강좌 3회차 - 멤버 접근을 제어하는 접근 지정자

1. 접근 지정자

기본적으로 구조체는 내부에 선언된 멤버 변수를 외부에서 제한 없이 접근이 가능합니다. 예를 들어, 아래의 예제에서 data 변수에 포함된 멤버 변수는 data.m_data 코드로 값을 저장할 수도 있고 읽을 수도 있습니다. 왜냐하면 C++ 언어는 구조체에 접근 지정자를 사용하지 않으면 모든 접근이 가능한 public 접근 지정자가 생략된 것으로 처리되기 때문입니다.

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

struct MyData
{
    int m_data;    // 멤버 변수 선언
};

int main()
{
    MyData data;

    data.m_data = 5;  // 멤버 변수에 5대입
    printf("%d\n", data.m_data);  // 멤버 변수에 저장된 값을 출력

    return 0;
}
```

위 예제는 아래와 같이 출력됩니다.

```
5

이 창을 닫으려면 아무 키나 누르세요...
```

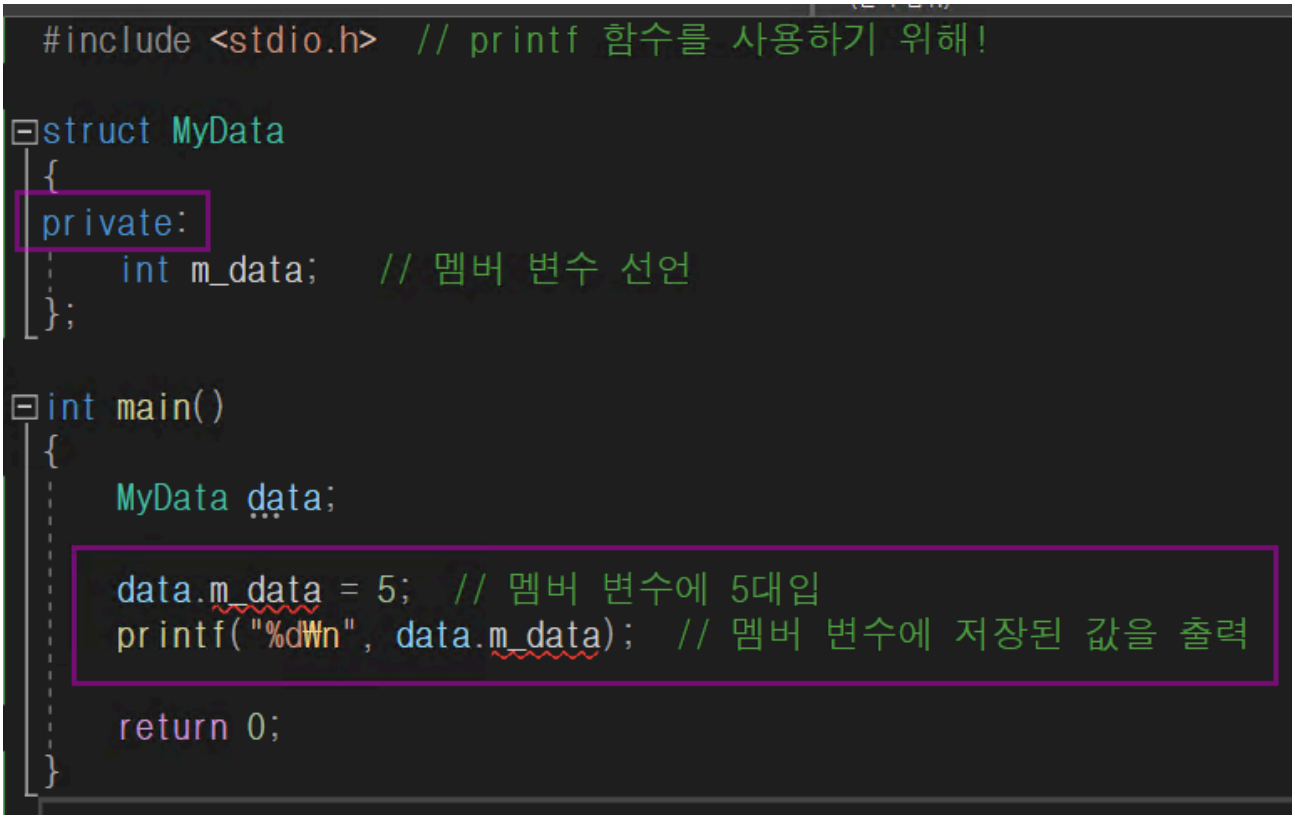
C++ 언어는 구조체 내부에 선언된 '멤버 변수'나 '멤버 함수'의 사용 권한을 부여하는 private, public 같은 '접근 지정자'가 제공됩니다. private, public 접근 지정자의 특성은 다음과 같고 여기서 사용이 불가능하다는 뜻은 사용하려고 하면 컴파일 할 때 문법 오류가 발생한다는 뜻입니다.

◆ private - 이 접근 지정자와 사용된 멤버는 구조체 외부에서 사용이 분가능
◆ public - 이 접근 지정자와 사용된 멤버는 구조체 외부에서 사용이 가능

접근 지정자를 사용하는 방법은 접근 지정자를 적고 ':'를 적은 다음에 멤버를 나열하면 됩니다. 예를 들어, 위에서 소개한 예제에서 MyData 구조체에 생략된 public 접근 지정자를 명시적으로 사용하게 적으면 다음과 같습니다.

```
struct MyData
{
public:
    int m_data;    // 멤버 변수 선언
};
```

그리고 구조체에 사용된 public 접근 지정자를 아래와 같이 private로 변경하면 멤버 변수를 사용하는 것이 문법적으로 제한되기 때문에 data.m_data라고 사용한 코드가 모두 오류 처리됩니다.



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

struct MyData
{
private:
    int m_data;   // 멤버 변수 선언
};

int main()
{
    MyData data;

    data.m_data = 5;  // 멤버 변수에 5대입
    printf("%d\n", data.m_data);  // 멤버 변수에 저장된 값을 출력

    return 0;
}
```

즉, 위 코드를 컴파일하면 아래와 같은 오류 메시지가 출력됩니다.

```
빌드 시작...
1>------ 빌드 시작: 프로젝트: ExamCPP, 구성: Debug Win32 ------
1>ExamCPP.cpp
1>error C2248: 'MyData::m_data': private 멤버('MyData' 클래스에서 선언)에 액세스할 수 없습니다.
1>message : 'MyData::m_data' 선언을 참조하십시오.
1>message : 'MyData' 선언을 참조하십시오.
1>error C2248: 'MyData::m_data': private 멤버('MyData' 클래스에서 선언)에 액세스할 수 없습니다.
1>message : 'MyData::m_data' 선언을 참조하십시오.
1>message : 'MyData' 선언을 참조하십시오.
1>"ExamCPP.vcxproj" 프로젝트를 빌드했습니다. - 실패
========== 빌드: 성공 0, 실패 1, 최신 0, 생략 0 ==========
```

만약, 이렇게 private 접근 지정자로 선언한 멤버 변수를 외부에서 사용하려면 아래와 같이 멤버 함수를 사용하면 됩니다. 물론 이 멤버 함수는 구조체 외부에서 사용해야 하기 때문에 public 접근 지정자에 추가해서 선언하면 됩니다. 클래스 내부에 있는 멤버들은 서로를 사용하는데 접근 지정자의 영향을 받지 않기 때문에 m_data가 private 접근 지정자로 선언되어도 SetData 함수나 GetData 함수에서 제한 없이 사용이 가능합니다. 그리고 main 함수에서 data.m_data 코드대신 data.SetData와 data.GetData 코드로 대체하면 컴파일 오류 없이 사용이 가능합니다.

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

struct MyData
{
private:
    int m_data;   // 멤버 변수 선언

public:
    // 외부에서 m_data 변수에 값을 저장할 때 사용하는 함수
    void SetData(int a_data)
    {
        m_data = a_data;
    }

    // 외부에서 m_data 변수에 저장된 값을 얻을 때 사용하는 함수
    int GetData()
    {
        return m_data;
    }
};

int main()
{
    MyData data;

    data.SetData(5);  // data.m_data = 5;와 동일하게 처리
    printf("%d\n", data.GetData()); // printf("%d\n", data.m_data);와 동일하게 처리

    return 0;
}
```

위 예제는 아래와 같이 출력됩니다.

```
5

이 창을 닫으려면 아무 키나 누르세요...
```

2. 접근 지정자를 왜 사용할까?

프로그래밍을 하다보면 아무리 계획을 잘해도 외부적인 요인(고객의 변심)이 있기 때문에 어쩔 수 없이 기능에 변화가 생기기 마련입니다. 그래서 이런 기능 변화가 생겼을 때 최소한의 코드만 변경해서 이 변화에 대처할 수 있도록 코드를 구성해야 하는데 이때 함수의 역할이 매우 큽니다.

하지만 개발자가 코드를 구성하면서 함수를 사용할 것인지 여부는 개발자의 선택사항입니다. 그래서 아직 유지보수에 대한 개념이 부족한 초보 개발자는 함수를 사용하지 않고 중복된 코드를 나열하는 경우가 많습니다. 예를 들어, 아래와 같이 함수

를 만드는데 익숙하지 않아서 코드를 나열하는 형태로 구성했는데 m_data 변수에 음수가 대입되면 양수로 변경해야 한다는 조건이 추가되면 어떻게 해야 할까요?

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

struct MyData
{
    int m_data;   // 멤버 변수 선언
};

int main()
{
    int temp;
    MyData data;

    scanf_s("%d", &temp);
    data.m_data = temp;

    /*
       ... 다른 코드 있음 ...
    */


    scanf_s("%d", &temp);
    data.m_data = temp;

    /*
       ... 다른 코드 있음 ...
    */

    scanf_s("%d", &temp);
    data.m_data = temp;

    return 0;
}
```

이렇게 함수처리가 되지 않은 코드는 어쩔수 없이 사용된 코드에 직접 변화에 대응하는 코드를 추가해야 합니다. 즉, 멤버 변수가 사용된 코드마다 기능 변화에 대처하는 코드를 추가해야 합니다.

```c
#include <stdio.h>  // printf, scanf_s 함수를 사용하기 위해!

struct MyData
{
    int m_data;   // 멤버 변수 선언
};

int main()
{
    int temp;
    MyData data;

    scanf_s("%d", &temp);
    if (temp < 0) temp = -temp;  // 사용자가 입력한 값이 음수면 양수로 변경
    data.m_data = temp;

    /*
       ... 다른 코드 있음 ...
    */


    scanf_s("%d", &temp);
    if (temp < 0) temp = -temp;  // 사용자가 입력한 값이 음수면 양수로 변경
    data.m_data = temp;

    /*
       ... 다른 코드 있음 ...
    */

    scanf_s("%d", &temp);
    if (temp < 0) temp = -temp;  // 사용자가 입력한 값이 음수면 양수로 변경
    data.m_data = temp;

    return 0;
}
```

그런데 C++ 언어를 배우는 초보자에게 구조체를 사용할 때 public이 아닌 private 접근 지정자를 사용하도록 권장했다면 m_data를 직접 사용하는 코드에 오류가 발생하기 때문에 아래와 같이 멤버 함수를 추가해서 m_data 멤버 변수를 사용하도록 코드를 구성했을 것입니다.

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

struct MyData
{
private:
    int m_data;   // 멤버 변수 선언

public:
    // 외부에서 m_data 변수에 값을 저장할 때 사용하는 함수
    void SetData(int a_data)
    {
        m_data = a_data;
    }
};

int main()
{
    int temp;
    MyData data;

    scanf_s("%d", &temp);
    data.SetData(temp);

    /*
       ... 다른 코드 있음 ...
    */


    scanf_s("%d", &temp);
    data.SetData(temp);

    /*
       ... 다른 코드 있음 ...
    */

    scanf_s("%d", &temp);
    data.SetData(temp);

    return 0;
}
```

그리고 위와 같이 작성된 코드에 m_data 변수에 음수가 대입되면 양수로 변경해야 한다는 조건이 추가되면 개발자는 아래와 같이 SetData 멤버 함수에 한 줄의 코드만 추가해서 이 변화에 대처할 수 있습니다. 즉, private 접근 지정자를 사용하도록 권장하면 함수를 사용하는 것이 선택 사항이 아니라 강제 사항으로 바뀌게 됩니다. 그래서 초보 개발자들이 자연스럽게 함수를 더 많이 만들어서 사용하게 되어 자신도 모르게 변화에 잘 대처하는 코드를 사용하게 됩니다.

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

struct MyData
{
private:
    int m_data;   // 멤버 변수 선언

public:
    // 외부에서 m_data 변수에 값을 저장할 때 사용하는 함수
    void SetData(int a_data)
    {
        if(a_data < 0) a_data = -a_data;  // 음수면 양수로 변경
        m_data = a_data;
    }
};

int main()
{
    int temp;
    MyData data;

    scanf_s("%d", &temp);
    data.SetData(temp);

    /*
       ... 다른 코드 있음 ...
    */


    scanf_s("%d", &temp);
    data.SetData(temp);

    /*
       ... 다른 코드 있음 ...
    */

    scanf_s("%d", &temp);
    data.SetData(temp);

    return 0;
}
```

3. 접근 지정자는 데이터를 보호할 수 있는가?

C++ 언어를 공부하다보면 접근 지정자를 설명하는 내용에 캡슐화(encapsulation) 개념을 함께 설명하는 경우가 많습니다. 여기서 캡슐화는 데이터를 보호하기 위해 특정 구조에 데이터를 은닉(정보 숨김, information hiding)하여 데이터를 보호하

는 개념입니다. 그런데 C++ 언어도 private 접근 지정자를 사용하면 멤버 변수를 구조체 외부에서 직접 접근하는 것이 불가능하기 때문에 구조체 내부에 선언한 멤버 변수가 은닉되어 데이터가 보호되는 효과가 있는 것처럼 강조합니다.

하지만 이것은 C, C++ 언어의 포인터 문법 때문에 지켜지지 않습니다. 즉, private 접근 지정자는 소스를 컴파일 할 때만 적용되는 규칙이기 때문에 실제 프로그램이 실행될 때는 적용되지 않습니다. 따라서 아래와 같이 **형 변환 연산자와 포인터 문법은 사용하면 private 접근 지정자가 지정된 멤버 변수의 값은 직접 수정하는 것이 가능**합니다.

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

struct MyData
{
private:
    int m_data;    // 멤버 변수 선언

public:
    // 외부에서 m_data 변수에 값을 저장할 때 사용하는 함수
    void SetData(int a_data)
    {
        if(a_data < 0) a_data = -a_data;  // 음수면 양수로 변경
        m_data = a_data;
    }

    // 외부에서 m_data 변수에 저장된 값을 얻을 때 사용하는 함수
    int GetData()
    {
        return m_data;
    }
};

int main()
{
    MyData data;

    *(int *)&data = 5; // data 변수에 있는 m_data 항목에 5가 대입 됨
    printf("%d\n", data.GetData());

    return 0;
}
```

위 예제는 아래와 같이 출력됩니다.

```
5

이 창을 닫으려면 아무 키나 누르세요...
```

따라서 private 키워드는 논리적인 보호(컴파일시에 체크)의 의미만 있을뿐 물리적인 보호(실행시에 메모리 접근)는 되지 않습니다. 결국 C++ 언어를 가르치는 사람들은 캡슐화까지 언급하며 데이터 보호의 중요성을 강조하면서까지 초보 개발자들이 private 키워드를 사용해야 하는 정당성을 주장합니다. 하지만 실제 private 키워드를 사용해야 하는 이유는 데이터의 보호보다는 초보 개발자들이 변화에 잘 대처하는 코드(함수)를 만드는 습관은 가지게 만드는데 목적이 있습니다.

여러분들은 이제 private 키워드의 목적을 배웠으니 좋은 프로그래밍 스타일을 가진 개발자가 되기 위해, 코드가 조금 길어져서 귀찮아지더라도 멤버 변수를 사용할 때는 반드시 private 키워드를 사용해야 합니다.

: 변화에 잘 대처하는 코드를 만드는 습관을 가지게 만드는데 목적이 있다!

2022.11.28. 21:33 답글쓰기

cpkjk 3

기록용 : (차후에 삭제할 것)
private:
int a = 4;
int b = 5;
};

Mydata v;
int tmp1, tmp2;
//tmp=*(char*)&v;
tmp1 = *(int*)&v;
//tmp2 = *((int*)&v+1);
tmp2=*((char*)&v+4);

2022.11.28. 21:35 답글쓰기

김성엽 M 작성자

ㅎㅎ 파이팅입니다!

2022.11.28. 23:05 답글쓰기

dh221009

댓글을 남겨보세요

등록