C++ 강좌 06회 : 생성자와 파괴자

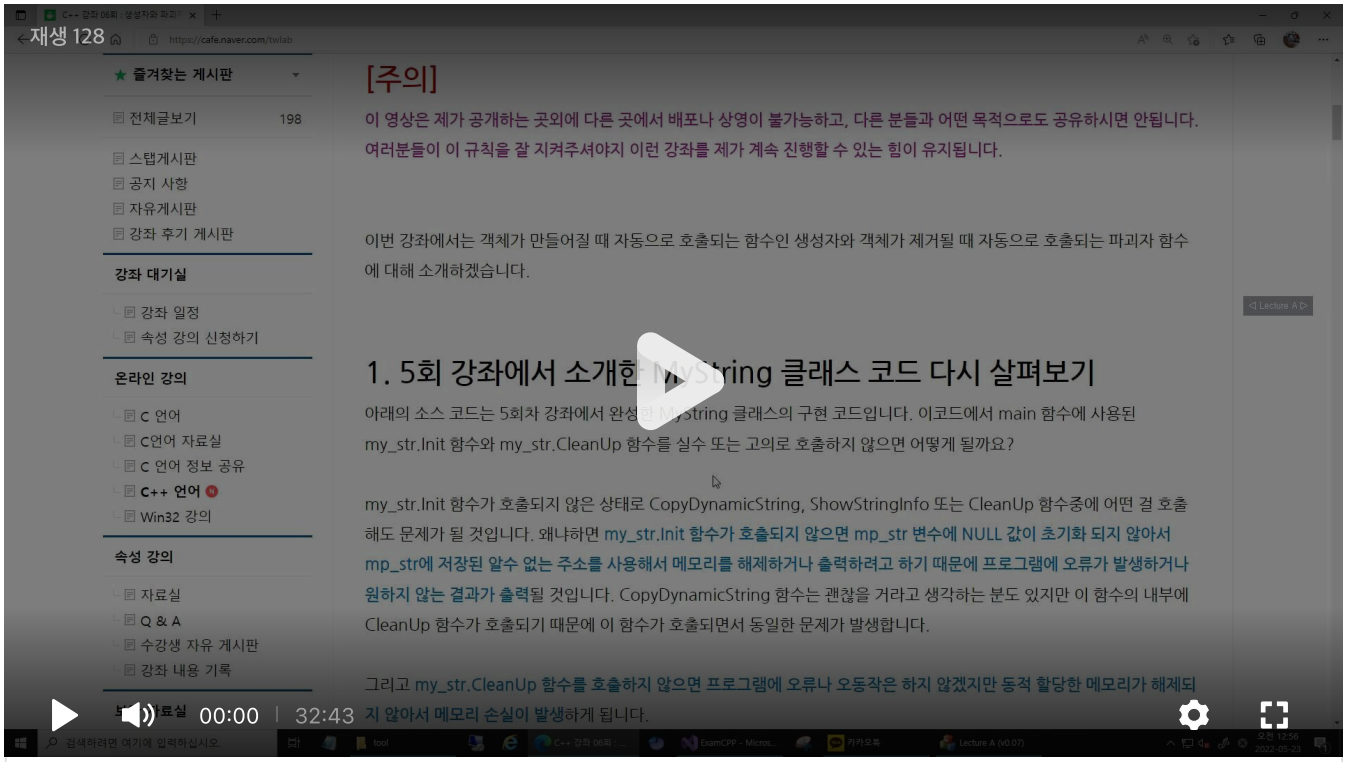
김성엽 카페매니저 + 구독 1:1 채팅

2022.05.22. 20:03 조회 323

[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 객체가 만들어질 때 자동으로 호출되는 함수인 생성자와 객체가 제거될 때 자동으로 호출되는 파괴자 함수에 대해 소개하겠습니다.



C++ 강좌 6회. 생성자와 파괴자

1. 5회 강좌에서 소개한 MyString 클래스 코드 다시 살펴보기

아래의 소스 코드는 5회차 강좌에서 완성한 MyString 클래스의 구현 코드입니다. 이코드에서 main 함수에 사용된 my_str.Init 함수와 my_str.CleanUp 함수를 실수 또는 고의로 호출하지 않으면 어떻게 될까요?

my_str.Init 함수가 호출되지 않은 상태로 CopyDynamicString, ShowStringInfo 또는 CleanUp 함수중에 어떤 걸 호출해도 문제가 될 것입니다. 왜냐하면 my_str.Init 함수가 호출되지 않으면 mp_str 변수에 NULL 값이 초기화 되지 않아서 mp_str에 저장된 알수 없는 주소를 사용해서 메모리를 해제하거나 출력하려고 하기 때문에 프로그램에 오류가 발생하거나 원하지 않는 결과가 출력될 것입니다. CopyDynamicString 함수는 괜찮을 거라고 생각하는 분도 있지만 이 함수의 내부에 CleanUp 함수가 호출되기 때문에 이 함수가 호출되면서 동일한 문제가 발생합니다.

그리고 my_str.CleanUp 함수를 호출하지 않으면 프로그램에 오류나 오동작은 하지 않겠지만 동적 할당한 메모리가 해제되지 않아서 메모리 손실이 발생하게 됩니다.

```
#include <stdio.h>    // printf, scanf_s 함수를 사용하기 위해
#include <string.h>   // strlen, memcpy 함수를 사용하기 위해
#include <malloc.h>   // malloc, free 함수를 사용하기 위해

class MyString
{
private:
    int m_len;  // NULL 문자를 포함한 문자열의 길이를 저장할 변수
    char *mp_str;  // 문자열이 저장된 메모리의 주소를 기억할 변수

public:
    // 멤버 변수를 초기화하는 함수
    void Init()
    {
        m_len = 0;
        mp_str = NULL;
    }

    // 동적메모리를 할당해서 a_str에 전달된 문자열을 복사하는 함수
    int CopyDynamicString(char a_str[])
    {
        // 이미 할당된 메모리가 있으면 제거한다.
        CleanUp();

        // 매개 변수로 전달된 문자열의 길이를 구한다.
        m_len = strlen(a_str) + 1;
        // 문자열을 저장하기 위해 동적 메모리를 할당한다.
        mp_str = (char *)malloc(m_len);
        // 메모리가 정상적으로 할당되었다면 메모리에 전달된 문자열을 복사한다.
        if (mp_str != NULL) memcpy(mp_str, a_str, m_len);
        // 문자열의 길이를 반환한다.
        return m_len;
    }

    // 이 객체가 관리하는 문자열 정보를 출력하는 함수
    void ShowStringInfo()
    {
        // 관리하는 문자열의 정보를 출력한다.
        if (mp_str != NULL) printf("%s : %d\n", mp_str, m_len);
        else printf("보관된 문자열이 없습니다.\n");
    }

    // 이 객체가 관리하던 정보를 정리하는 함수
    void CleanUp()
    {
        // 할당된 메모리가 있다면 해제한다.
        if (mp_str != NULL) {
            free(mp_str);
            mp_str = NULL;
        }
    }
};
```

```
int main()
{
    char str[32];
    // 사용자에게 32자 이하의 문자열을 입력 받는다.
    scanf_s("%s", str, 32);

    MyString my_str; // 문자열 정보를 저장할 객체
    my_str.Init();   // 객체를 초기화한다.

    // 동적 메모리를 할당하고 str에 저장된 문자열을 복사
    my_str.CopyDynamicString(str);
    // 문자열이 잘 복사되었는지 확인한다.
    my_str.ShowStringInfo();
    // 객체가 사용하던 정보를 정리한다.
    my_str.CleanUp();
    return 0;
}
```

결국 위 예제에서 **main 함수에 사용된 my_str.Init(); 코드와 my_str.CleanUp(); 코드는 선택이 아니라 객체가 생성되면 반드시 사용해야 하는 코드**입니다.

2. 필수 코드를 개발자가 적게 하는 것이 올바른 방법일까?

객체는 행위를 강조하는 개념이라서 데이터 형식이 행위(함수)에 숨겨진 상태로 제공됩니다. 따라서 객체 내부에 선언된 멤버 변수를 초기화하거나 정리하는 작업이 필요한 경우가 많습니다. 물론, 객체를 초기화하거나 정리하는 함수가 반드시 제공되어야 하는 것은 아니지만, 객체를 만든 개발자가 **객체를 초기화하는 함수나 정리하는 함수를 제공했다면 객체를 사용하는 개발자는 이 함수를 반드시 사용**해야 합니다.

예를 들어, 위 예제에서 MyString 클래스가 객체를 초기화하는 Init 함수와 객체를 정리하는 CleanUp 함수를 제공하기 때문에, my_str 객체를 만들었다면 다른 함수를 호출하기 전에 my_str.Init(); 코드를 반드시 호출해야 합니다. 그리고 객체를 사용하다가 객체가 제거되는 시점(지역 변수는 함수가 종료될 때 제거 됨)에 my_str.CleanUp(); 코드를 반드시 사용해서 사용하던 메모리를 정리해야 합니다.

결국 **MyString 클래스를 구현하는 개발자가 Init 함수와 CleanUp 함수를 추가하는 순간에 이 객체는 두 함수가 호출되어야 한다는 필수 사항이 생기는 것**입니다. 그래서 클래스를 구현한 개발자는 이 클래스를 사용할 개발자에게 이 두 함수의 존재를 알리기 위해 클래스 사용법에 주의 사항까지 추가해야 합니다. 이런 주의 사항이 추가된다는 뜻은 이 클래스를 사용할 개발자가 코드 구현에 어려움을 느끼게 될 것이라는 뜻이기도 합니다. 즉, 클래스를 만드는 개발자는 불편한 자료나 교육을 해야하는 불편함이 생기고 클래스를 사용하는 개발자는 이 클래스에 대한 더 많은 이해와 경험을 강요받게 됩니다.

그리고 이런 필수 상황은 클래스를 사용할 개발자에게 아무리 강조해도 Init 함수나 CleanUp 함수를 실수로 사용하지 않아 문제가 발생하게 됩니다. 즉, 클래스 사용의 문제를 막기위해 노력하는데도 문제를 완벽하게 막을수 없다는 불편한 진실이 존재하는 것입니다.

```
"소스에서 필수적으로 해야 하는 것을 개발자가 적게하면 반드시 실수하는 경우가 생깁니다."
```

3. 생성자와 파괴자

개발자가 C++ 언어로 작성한 소스 코드는 컴파일러에 의해 기계어로 번역되어 실행 파일에 저장됩니다. 따라서 개발자가 필수 코드를 실수로 적지 않아도 컴파일러가 추가해 줄 수 있는 기회가 있다는 뜻입니다. 그리고 이렇게 **필수 코드를 추가해 주는 기능은 컴파일러의 기능에 포함할 거라면 필수 코드 자체를 개발자가 적은 필요가 없게 만드는 것이 더 좋은 것**입니다.

클래스를 만드는 개발자가 객체를 초기화하거나 정리하는 함수를 만들었다면 해당 함수는 반드시 사용해야 한다는 뜻입니다. 따라서 **컴파일러는 C++ 언어로 작성된 소스 코드를 기계어로 번역하는 작업에서, 클래스가 사용되어 객체가 만들어지면 객체를 초기화하거나 정리하는 코드를 자동으로 추가**하면 됩니다. 이렇게 하면 클래스를 만드는 사람은 이 클래스를 사용할 때 필수로 해야할 작업들을 사용자에게 알려야 할 필요가 없고 클래스를 사용하는 개발자는 이런 코드를 적지 않아서 작업이 편해지고 쉬워질 것입니다. 그리고 필수로 적어야 하는 코드를 적지 않아서 발생하는 문제도 사라질 것입니다.

하지만 컴파일러 개발자는 이 작업을 구현하는데 어려움이 생겼을 것입니다. 왜냐하면 함수의 이름은 개발자가 원하는대로 작성되기 때문에 컴파일러 입장에서는 클래스의 어떤 함수가 객체를 초기화하는 함수이고 객체를 정리하는 함수인지 예상할 수 없기 때문입니다. 따라서 객체를 초기화하는 함수와 객체를 정리하는 함수에 대한 이름 규칙을 문법으로 제공하는데 이 문법이 '생성자'와 '파괴자'입니다.

'생성자'는 객체가 만들어졌을 때 객체를 초기화하기 위해 자동으로 호출되는 함수입니다. 그리고 **'파괴자'는 객체가 사라질 때 객체가 사용하던 내부 정보를 정리하기 위해 자동으로 호출되는 함수**입니다. 생성자는 클래스와 이름이 같은 함수이며 파괴자는 클래스와 이름이 같으면서 앞에 ~기호가 추가된 함수입니다. 그리고 이 두 함수는 자동으로 호출되기 때문에 **소스 코드에 코드가 추가되는 것이 아니라서 반환 값을 사용할 수 없습니다.** 그래서 반환 자료형도 없고 return 문으로 값을 반환 할 수도 없습니다.

예를 들어, MyString 클래스에 생성자와 파괴자 개념을 적용해서 Init 함수는 생성자 함수로 변경하고 CleanUp 함수는 CopyDynamicString 함수에서도 사용하기 때문에 파괴자 함수를 추가해서 CleanUp 함수를 호출하게 아래와 같이 변경했습니다. 그리고 생성자와 파괴자는 자동으로 호출되기 때문에 main 함수 코드에서 Init 함수와 CleanUp 함수를 호출하는 코드가 사라졌습니다.

```
#include <stdio.h>    // printf, scanf_s 함수를 사용하기 위해
#include <string.h>   // strlen, memcpy 함수를 사용하기 위해
#include <malloc.h>   // malloc, free 함수를 사용하기 위해

class MyString
{
private:
    int m_len;  // NULL 문자를 포함한 문자열의 길이를 저장할 변수
    char *mp_str;  // 문자열이 저장된 메모리의 주소를 기억할 변수

public:
    // 생성자 : 객체를 초기화하기 위해 자동으로 호출되는 함수
    MyString()
    {
        m_len = 0;
        mp_str = NULL;
    }

    // 파괴자 : 객체를 정리하기 위해 자동으로 호출되는 함수
    ~MyString()
    {
        CleanUp();
    }

    // 동적메모리를 할당해서 a_str에 전달된 문자열을 복사하는 함수
    int CopyDynamicString(char a_str[])
    {
        // 이미 할당된 메모리가 있으면 제거한다.
        CleanUp();

        // 매개 변수로 전달된 문자열의 길이를 구한다.
        m_len = strlen(a_str) + 1;
        // 문자열을 저장하기 위해 동적 메모리를 할당한다.
        mp_str = (char *)malloc(m_len);
        // 메모리가 정상적으로 할당되었다면 메모리에 전달된 문자열을 복사한다.
        if (mp_str != NULL) memcpy(mp_str, a_str, m_len);
        // 문자열의 길이를 반환한다.
        return m_len;
    }

    // 이 객체가 관리하는 문자열 정보를 출력하는 함수
    void ShowStringInfo()
    {
        // 관리하는 문자열의 정보를 출력한다.
        if (mp_str != NULL) printf("%s : %d\n", mp_str, m_len);
        else printf("보관된 문자열이 없습니다.\n");
    }

    // 이 객체가 관리하던 정보를 정리하는 함수
    void CleanUp()
    {
        // 할당된 메모리가 있다면 해제한다.
```

```
        if (mp_str != NULL) {
            free(mp_str);
            mp_str = NULL;
        }
    }
};

int main()
{
    char str[32];
    // 사용자에게 32자 이하의 문자열을 입력 받는다.
    scanf_s("%s", str, 32);

    MyString my_str; // 문자열 정보를 저장할 객체

    // 동적 메모리를 할당하고 str에 저장된 문자열을 복사
    my_str.CopyDynamicString(str);
    // 문자열이 잘 복사되었는지 확인한다.
    my_str.ShowStringInfo();

    return 0;
}
```

결국, 클래스 문법에 생성자와 파괴자 문법이 추가되어, 클래스를 사용하는 개발자 입장에서는 사용해야 할 코드가 줄어 코드 작성이 더 편해지고 필수 코드와 관련된 오듀도 발생하지 않게 됩니다.

4. 메모리 할당과 객체의 인스턴스

아래와 같이 자료형을 사용해서 변수를 선언하면 '변수를 메모리에 할당(allocation)한다'고 이야기 합니다. 왜냐하면 아래의 코드는 메모리에 4바이트 크기의 메모리를 정수 형식으로 사용하겠다는 메모리 할당의 의미를 가지고 있기 때문입니다.

```
int data;   // 변수 할당
```

그런데 아래와 같이 MyString 클래스로 my_str 객체를 선언하는 것은 할당이라는 표현을 사용하지 않습니다. 왜냐하면 아래의 표현은 메모리 할당 작업과 유사하지만 생성자 호출 코드가 포함되어 있기 때문에 '할당 + 객체 초기화 함수 호출'의 의미입니다. 따라서 이 작업을 할당 작업과 구분짓기 위해 C++ 언어에서는 '객체를 인스턴스(instance)한다.'고 합니다.

```
MyString my_str;  // 객체 인스턴스
// my_str.MyString(); 함수가 자동으로 호출 됨
```

물론, instance라는 단어가 '예시'라는 뜻도 있습니다. 그래서 클래스의 사용 예시가 객체이기 때문에 instance 용어의 의미를 객체라고 확대 해석하는 경우도 있습니다. 하지만 instance 용어는 다양한 기술에서 좀더 포괄적인 의미로 사용되기 때문에 객체를 instance로 해석하는 것은 오해의 여지가 있습니다. 따라서 객체는 그냥 객체로 이야기하고 객체가 만들어지는 행위를 instance로 이야기하는 것이 좋습니다.

5. 생성자와 파괴자의 사용 조건

클래스를 만들면서 생성자 함수와 파괴자 함수를 반드시 만들 필요는 없습니다. 클래스를 만드는 개발자가 판단해서 필요하면 추가하고 필요하지 않으면 생략해도 됩니다. 그리고 당연히 생략했다면 자동으로 호출되는 기능은 무시됩니다.

생성자의 경우에는 객체를 instance 할 때 함께 호출되기 때문에 객체 이름 뒤에 괄호를 사용해서 매개 변수 사용에 대한 표현을 추가할 수 있습니다. 따라서 **생성자는 매개 변수를 가지는 것이 가능하기 때문에 한 개의 클래스에 여러 개의 생성자를 선언하는 것이 가능**합니다. 생성자를 여러 개 추가한 경우 **매개 변수가 없는 생성자를 '기본 생성자'**라고 합니다.

```
MyString my_str;  // 매개 변수가 없는 생성자(기본 생성자)를 호출한다.
MyString my_str("abc");  // 문자열 상수를 매개 변수로 가지는 생성자를 호출한다.
```

예를 들어, 위와 같이 사용 가능하게 하려면 아래와 같이 MyString 클래스에 생성자를 한 개더 추가해주면 됩니다.

```
#include <stdio.h>    // printf, scanf_s 함수를 사용하기 위해
#include <string.h>   // strlen, memcpy 함수를 사용하기 위해
#include <malloc.h>   // malloc, free 함수를 사용하기 위해

class MyString
{
private:
    int m_len;  // NULL 문자를 포함한 문자열의 길이를 저장할 변수
    char *mp_str;  // 문자열이 저장된 메모리의 주소를 기억할 변수

public:
    // 기본 생성자 : 객체를 초기화하기 위해 자동으로 호출되는 함수
    MyString()
    {
        m_len = 0;
        mp_str = NULL;
    }

    // 생성자 : 지정된 문자열을 사용해서 객체를 초기화하는 생성자 함수
    MyString(const char *ap_str)
    {
        mp_str = NULL;
        CopyDynamicString(ap_str);
    }

    // 파괴자 : 객체를 정리하기 위해 자동으로 호출되는 함수
    ~MyString()
    {
        CleanUp();
    }

    // 동적메모리를 할당해서 a_str에 전달된 문자열을 복사하는 함수
    int CopyDynamicString(const char a_str[])
    {
        // 이미 할당된 메모리가 있으면 제거한다.
        CleanUp();

        // 매개 변수로 전달된 문자열의 길이를 구한다.
        m_len = strlen(a_str) + 1;
        // 문자열을 저장하기 위해 동적 메모리를 할당한다.
        mp_str = (char *)malloc(m_len);
        // 메모리가 정상적으로 할당되었다면 메모리에 전달된 문자열을 복사한다.
        if (mp_str != NULL) memcpy(mp_str, a_str, m_len);
        // 문자열의 길이를 반환한다.
        return m_len;
    }

    // 이 객체가 관리하는 문자열 정보를 출력하는 함수
    void ShowStringInfo()
    {
        // 관리하는 문자열의 정보를 출력한다.
        if (mp_str != NULL) printf("%s : %d\n", mp_str, m_len);
```

```
        else printf("보관된 문자열이 없습니다.\n");
    }

    // 이 객체가 관리하던 정보를 정리하는 함수
    void CleanUp()
    {
        // 할당된 메모리가 있다면 해제한다.
        if (mp_str != NULL) {
            free(mp_str);
            mp_str = NULL;
        }
    }
};

int main()
{
    char str[32];
    // 사용자에게 32자 이하의 문자열을 입력 받는다.
    scanf_s("%s", str, 32);

    MyString my_str; // 문자열 정보를 저장할 객체 (기본 생성자 사용)
    // my_str.MyString(); 호출
    // 동적 메모리를 할당하고 str에 저장된 문자열을 복사
    my_str.CopyDynamicString(str);
    // 문자열이 잘 복사되었는지 확인한다.
    my_str.ShowStringInfo();

    // 문자열 상수를 인자로 가지는 생성자 사용
    MyString temp_str("tipsware lab");
    // temp_str.MyString("tipsware lab"); 호출
    // 문자열이 잘 복사되었는지 확인한다.
    temp_str.ShowStringInfo();

    return 0;
}
```

이처럼 **생성자 함수는 필요하지 않다면 생략하는 것도 가능하고 한 개 이상은 추가하는 것도 가능**합니다. 하지만 파괴자 함수는 별도로 호출을 의미하는 코드를 사용하지 않기 때문에 매개 변수를 전달하는 코드를 적을 수 있는 코드가 없습니다. 따라서 **파괴자 함수는 필요하지 않다면 생략하는 것은 가능하지만 필요하다면 반드시 한 개만 사용이 가능하고 매개 변수는 사용할 수 없습니다.**


수고하셨습니다

2022.07.07. 23:19 답글쓰기

황금잉어가물치 3

1번 예제에서 class내부의 int CopyDynamicString함수 내부의 Cleanup은 사진과 같이 "CleanUp(this)"의 this가 생략이 된 것인지 궁금합니다.

2022.11.12. 18:55 답글쓰기

김성엽 M 작성자

의미적으로 생략된거라고 할수 있습니다. 생략이라는 뜻은 다시 써도 된다는 뜻을 포함한다는 의미도 가지고 있어서 오해할수도 있을듯하네요. 우리가 직접 생략한것은 아니지만 언어적으로 생략되어 있는거라고 보시면 됩니다.

2022.11.12. 19:38 답글쓰기

황금잉어가물치 3

김성엽 감사합니다

2022.11.13. 02:02 답글쓰기

카일 3

강의 잘 들었습니다. 용어 정의 복습하고 갑니다~

2022.11.19. 07:04 답글쓰기

김성엽 M 작성자

파이팅입니다 ㅎ

2022.11.19. 14:44 답글쓰기

dh221009

댓글을 남겨보세요

등록