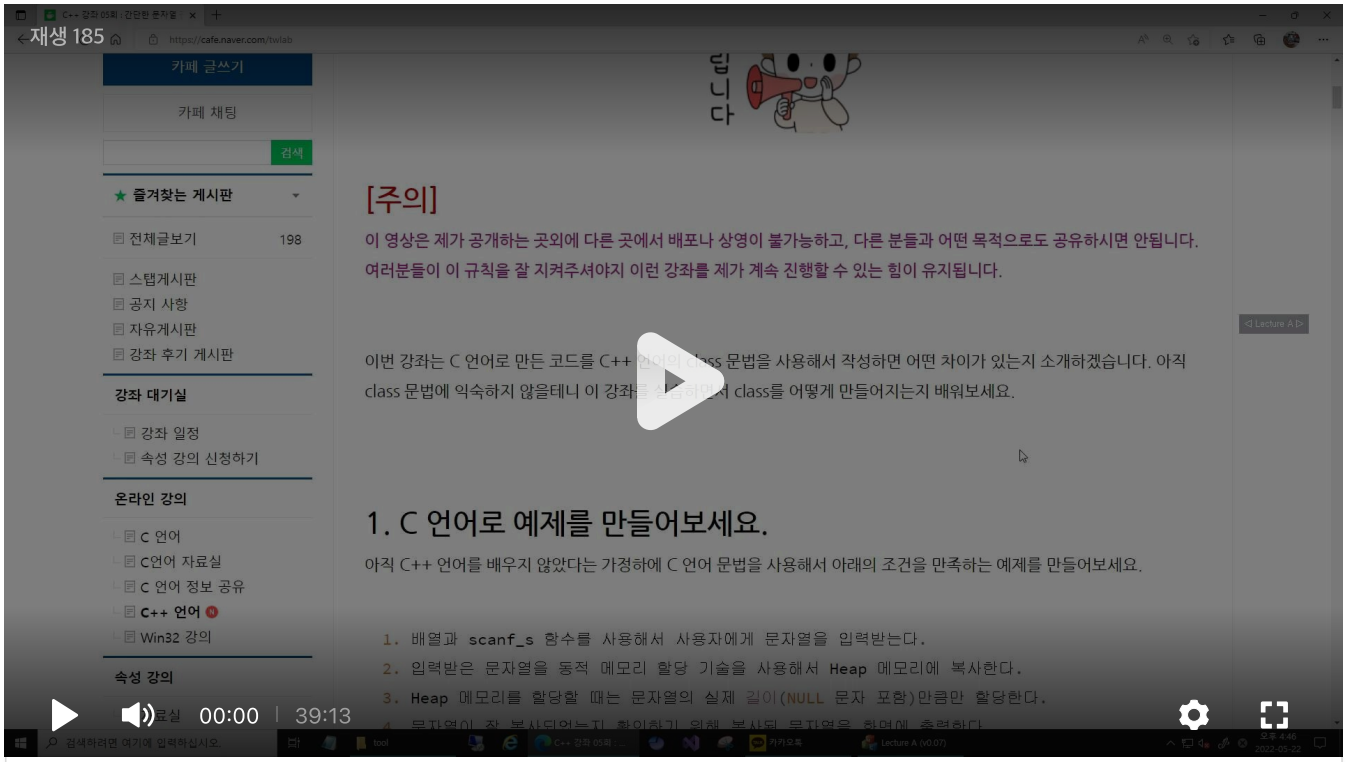
C++ 강좌 05회 : 간단한 문자열 관리 클래스 만들기

김성엽 카페매니저 + 구독 1:1 채팅
2022.05.21. 00:35 조회 439

[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 강좌는 C 언어로 만든 코드를 C++ 언어의 class 문법을 사용해서 작성하면 어떤 차이가 있는지 소개하겠습니다. 아직 class 문법에 익숙하지 않을테니 이 강좌를 실습하면서 class를 어떻게 만들어지는지 배워보세요.



5회. 간단한 문자열 관리 클래스 만들기

1. C 언어로 예제를 만들어보세요.

아직 C++ 언어를 배우지 않았다는 가정하에 C 언어 문법을 사용해서 아래의 조건을 만족하는 예제를 만들어보세요.

1. 배열과 scanf_s 함수를 사용해서 사용자에게 문자열을 입력받는다.
2. 입력받은 문자열을 동적 메모리 할당 기술을 사용해서 Heap 메모리에 복사한다.
3. Heap 메모리를 할당할 때는 문자열의 실제 길이(NULL 문자 포함)만큼만 할당한다.
4. 문자열이 잘 복사되었는지 확인하기 위해 복사된 문자열을 화면에 출력한다.
5. 동적할당된 메모리를 해제한다.

직접 만들어 보셨나요?

일단 가장 일반적인 답안을 소개하면 다음과 같습니다.

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

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

    // 입력받은 문자열의 길이를 구한다.
    int len = strlen(str);
    // 입력받은 문자열을 저장하기 위해 동적 메모리를 할당한다.
    char *p_str = (char *)malloc(len + 1);
    if (p_str != NULL) {
        // 할당된 메모리에 사용자가 입력한 문자열을 복사한다.
        strcpy_s(p_str, len + 1, str);
        // 잘 복사되었는지 확인한다.
        printf("%s\n", p_str);

        // 할당된 메모리를 해제한다.
        free(p_str);
    }
    return 0;
}
```

위 예제는 아래와 같이 출력됩니다.

```
tipsware
tipsware

이 창을 닫으려면 아무 키나 누르세요...
```

위 코드를 조금 더 효율적인 코드로 수정한다면 아래와 같이 해도 됩니다. len + 1가 반복적으로 사용되기 때문에 strlen 함수에서 값은 받은 때 1을 더해서 len은 저장해두면 len + 1을 반복할 필요가 없습니다. 그리고 문자열의 길이를 구한 상태이기 때문에 문자열을 복사할 때 strcpy_s 함수대신 memcpy 함수를 사용하는 것이 더 좋습니다.

```c
#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

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

    // 입력받은 문자열의 길이를 구한다. (NULL 문자 포함 길이)
    int len = strlen(str) + 1;
    // 입력받은 문자열을 저장하기 위해 동적 메모리를 할당한다.
    char *p_str = (char *)malloc(len);
    if (p_str != NULL) {
        // 할당된 메모리에 사용자가 입력한 문자열을 복사한다.
        memcpy(p_str, str, len);
        // 잘 복사되었는지 확인한다.
        printf("%s\n", p_str);

        // 할당된 메모리를 해제한다.
        free(p_str);
    }
    return 0;
}
```

2. 사용자 함수를 추가해서 코드를 재구성 합시다.

위 예제를 만들어야 하는데, 작업을 진행 할 개발자가 포인터는 배웠지만 동적 메모리 할당을 배우지 못했다면 어떻게 해야 할까요?

가장 좋은 방법은 해당 개발자가 동적 메모리 할당을 공부해서 스스로 작업할 수 있을 때까지 기다려 주는 것입니다. 하지만 회사에서 당장 작업을 진행해야 하는데 함께 일하는 동료가 기다려 줄 수 있을까요?

그래서 실력있는 개발자들은 함께 일할 초보자들을 위해, 동적 메모리 할당에 대한 이해없이 위 작업을 구현할 수 있도록 아래와 같이 CopyDynamicString, FreeDynamicString 함수를 미리 만들어서 제공하는 방법을 선택하기도 합니다. 이렇게 하면 **초보 개발자들은 동적 메모리 할당에 대해 공부하지 않고도 CopyDynamicString, FreeDynamicString 함수의 사용법만 익혀서 원하는 작업을 할 수 있습니다.** 만약, 이 두 함수의 코드를 노출하기 싫거나 초보 개발자들이 함수의 내부 코드를 보는 것만으로도 부담을 느낄수 있다면 이런 함수들을 라이브러리로 구성해서 제공하면 됩니다.

```c
#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

// 동적메모리를 할당해서 a_str에 전달된 문자열을 복사하는 함수
char *CopyDynamicString(char a_str[])
{
    // 매개 변수로 전달된 문자열의 길이를 구한다.
    int len = strlen(a_str) + 1;
    // 문자열을 저장하기 위해 동적 메모리를 할당한다.
    char *p_str = (char *)malloc(len);
    // 메모리가 정상적으로 할당되었다면 메모리에 전달된 문자열을 복사한다.
    if (p_str != NULL) memcpy(p_str, a_str, len);
    // 할당된 메모리의 주소를 반환한다.
    return p_str;
}

// 할당된 메모리를 해제하는 함수
void FreeDynamicString(char *ap_str)
{
    // 할당된 메모리가 있다면 해제한다.
    if (ap_str != NULL) free(ap_str);
}

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

    // 동적 메모리를 할당하고 str에 저장된 문자열을 복사한 다음 해당 주소를 반환한다.
    char *p_str = CopyDynamicString(str);

    // 문자열이 잘 복사되었는지 확인한다.
    if (p_str != NULL) printf("%s\n", p_str);

    // 사용하던 동적 메모리를 해제한다.
    FreeDynamicString(p_str);
    return 0;
}
```

위 예제에서 main 함수를 보면 동일한 기능은 구현되었지만 동적 메모리 할당과 관련된 내용이 사라져서 기존 코드보다 main 함수가 보기 더 쉬워졌을 겁니다. 물론 CopyDynamicString 함수는 문자열을 복사할 때 사용하고 FreeDynamicString 함수는 사용하던 문자열을 제거할 때 사용한다는 정도는 공부해야 합니다.

3. 문자열 길이 정보를 함께 사용해야 한다면?

보통 함수를 만들어서 사용하게 되면 작업 과정에 사용되는 정보들은 함수 내부에 숨겨지게 됩니다. 그렇기 때문에 사용하는 코드가 단순해져서 함수를 사용하는 개발자가 편하게 느끼는 것입니다. 하지만 그 숨겨진 정보가 필요하게 되면 어떻게 될까요? 예를 들어, **복사된 문자열의 길이 정보(NULL 문자 포함 길이)를 main 함수에서 사용해야 한다**면 어떻게 해야 할까요?

CopyDynamicString 함수를 만든 사람은 함수 내부에 사용된 문자열 길이를 main 함수에 넘기기 위해 아래와 같이 포인터 형식의 매개 변수를 추가해서 이 정보를 사용할 수 있게 해줘야 합니다.

```c
#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

// 동적메모리를 할당해서 a_str에 전달된 문자열을 복사하는 함수
char *CopyDynamicString(char a_str[], int *ap_len)
{
    // 매개 변수로 전달된 문자열의 길이를 구한다.
    *ap_len = strlen(a_str) + 1;
    // 문자열을 저장하기 위해 동적 메모리를 할당한다.
    char *p_str = (char *)malloc(*ap_len);
    // 메모리가 정상적으로 할당되었다면 메모리에 전달된 문자열을 복사한다.
    if (p_str != NULL) memcpy(p_str, a_str, *ap_len);
    // 할당된 메모리의 주소를 반환한다.
    return p_str;
}

// 할당된 메모리를 해제하는 함수
void FreeDynamicString(char *ap_str)
{
    // 할당된 메모리가 있다면 해제한다.
    if (ap_str != NULL) free(ap_str);
}

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

    // 문자열의 길이를 저장할 변수 (NULL 문자 포함)
    int len = 0;
    // 동적 메모리를 할당하고 str에 저장된 문자열을 복사한 다음 해당 주소를 반환한다.
    char *p_str = CopyDynamicString(str, &len);

    // 문자열이 잘 복사되었는지 확인한다.
    if (p_str != NULL) printf("%s : %d\n", p_str, len);

    // 사용하던 동적 메모리를 해제한다.
    FreeDynamicString(p_str);
    return 0;
}
```

위 예제는 아래와 같이 출력됩니다.

```
tipsware
tipsware : 9

이 창을 닫으려면 아무 키나 누르세요...
```

만약, 문자열 길이를 매개 변수로 가져오지 않고 반환 값을 사용해서 가져온다면 아래와 같이 매개 변수에 문자열이 저장된 주소를 가져올 수 있도록 2차 포인터를 사용해야 하기 때문에 코드는 더 복잡해 집니다.

```
#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

// 동적메모리를 할당해서 a_str에 전달된 문자열을 복사하는 함수
int CopyDynamicString(char **ap_dest_str, char a_str[])
{
    // 매개 변수로 전달된 문자열의 길이를 구한다.
    int len = strlen(a_str) + 1;
    // 문자열을 저장하기 위해 동적 메모리를 할당한다.
    *ap_dest_str = (char *)malloc(len);
    // 메모리가 정상적으로 할당되었다면 메모리에 전달된 문자열을 복사한다.
    if (*ap_dest_str != NULL) memcpy(*ap_dest_str, a_str, len);
    // 문자열의 길이를 반환한다.
    return len;
}

// 할당된 메모리를 해제하는 함수
void FreeDynamicString(char *ap_str)
{
    // 할당된 메모리가 있다면 해제한다.
    if (ap_str != NULL) free(ap_str);
}

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

    // 문자열이 할당된 메모리의 주소를 저장할 변수
    char *p_str = NULL;
    // 동적 메모리를 할당하고 str에 저장된 문자열을 복사한 다음
    // 해당 메모리의 주소를 p_str에 저장하고 문자열의 길이를 반환한다.
    int len = CopyDynamicString(&p_str, str);

    // 문자열이 잘 복사되었는지 확인한다.
    if (p_str != NULL) printf("%s : %d\n", p_str, len);

    // 사용하던 동적 메모리를 해제한다.
    FreeDynamicString(p_str);
    return 0;
}
```

위 두 가지 방법 중에 어떤 방법을 사용하든 함수 내부 코드와 함수를 사용하는 코드가 다 복잡해지고 사용법도 어려워지게 됩니다. 이렇게 되면 이런 함수를 사용하는 것이 과연 더 좋은 방법인지 고민하게 될 것입니다. 하지만 초보 개발자가 동적 메모리 할당을 실수하는 사건이 몇 번 발생하면 다시 함수를 제공하는 방법이 더 좋다는 결론이 나올 것입니다.

그런데 이 선택에는 문제가 있습니다. **함수에 전달되는 매개 변수나 반환 값에 변화가 생기면 CopyDynamicString 함수를 만든 사람뿐만 아니라 이 함수를 main 함수에 사용하는 개발자도 함께 코드를 변경**해야 합니다. 즉, 이렇게 데이터 사용 조건이 변경되면 프로그램 전반에 변화가 생기게 되어 코드 유지 보수가 어려워 진다는 문제를 해결해야 합니다.

4. 구조체 문법을 사용해서 해결하면 됩니다.

CopyDynamicString 함수가 데이터 사용 조건 변화에 대처를 못하게 된 문제는 함수 문법이 가지는 단점이라기 보다는 이 **함수를 만든 개발자가 문자열의 주소와 문자열의 길이가 하나의 정보라는 것을 예상하지 못해서 생긴 문제**일 뿐입니다. 즉, 함수가 좀더 변화에 잘 대처하는 구조를 가지려면 함수가 제공하는 정보의 범위를 미리 예상하고 그 변화에 대처할 수 있도록 아래와 같이 제공할 정보를 구조체로 정의해서 사용해야 합니다.

```
struct MyString
{
    int len;  // NULL 문자를 포함한 문자열의 길이를 저장할 변수
    char *p_str;  // 문자열이 저장된 메모리의 주소를 기억할 변수
};
```

이렇게 구조체를 사용하게 되면, 문자열 구성 정보에 추가적인 요구 사항이 생기더라도 매개 변수에 나열되지 않고 MyString 구조체에 해당 데이터가 포함될 것입니다. 그래서 변화 요소가 생겨도 변화에 따른 코드 변경이 **MyString 구조체와 CopyDynamicString 함수 내부 코드를 수정하는 것으로 대응이 가능**해집니다. 즉, main 함수에서 CopyDynamicString 함수를 호출하는 코드에는 변화가 생기지 않게 구성할 수 있습니다.

MyString 구조체를 사용하도록 예제를 수정해보면 다음과 같습니다.

```c
#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

struct MyString
{
    int len;  // NULL 문자를 포함한 문자열의 길이를 저장할 변수
    char *p_str;  // 문자열이 저장된 메모리의 주소를 기억할 변수
};

// 동적메모리를 할당해서 a_str에 전달된 문자열을 복사하는 함수
int CopyDynamicString(MyString *ap_my_str, char a_str[])
{
    // 매개 변수로 전달된 문자열의 길이를 구한다.
    ap_my_str->len = strlen(a_str) + 1;
    // 문자열을 저장하기 위해 동적 메모리를 할당한다.
    ap_my_str->p_str = (char *)malloc(ap_my_str->len);
    // 메모리가 정상적으로 할당되었다면 메모리에 전달된 문자열을 복사한다.
    if (ap_my_str->p_str != NULL) memcpy(ap_my_str->p_str, a_str, ap_my_str->len);
    // 문자열의 길이를 반환한다.
    return ap_my_str->len;
}

// 할당된 메모리를 해제하는 함수
void FreeDynamicString(MyString *ap_my_str)
{
    // 할당된 메모리가 있다면 해제한다.
    if (ap_my_str->p_str != NULL) free(ap_my_str->p_str);
}

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

    // 문자열 정보를 저장할 구조체 변수
    MyString my_str = { 0, NULL };
    // 동적 메모리를 할당하고 str에 저장된 문자열을 복사
    CopyDynamicString(&my_str, str);

    // 문자열이 잘 복사되었는지 확인한다.
    if (my_str.p_str != NULL) printf("%s : %d\n", my_str.p_str, my_str.len);

    // 사용하던 동적 메모리를 해제한다.
    FreeDynamicString(&my_str);
    return 0;
}
```

위 예제 코드를 보면 알겠지만, 제가 1회차 강좌에서 소개했던것처럼 C 언어는 구조체와 함수 그리고 포인터를 사용하는 문법으로 구조가 만들어지고 작업이 진행된다는 것을 알 수 있을 겁니다. 그렇다면 이 코드를 C++ 언어에서 객체 개념을 제공하는 class 문법으로 재구성하면 어떻게 될까요?

5. class 문법은 사용하면 코드가 어떻게 달라지는가?

먼저 아래와 같이 struct 문법으로 정의했던 MyString을 class 문법으로 수정해야 합니다. 제가 권장했듯이 접근 지정자는 private로 지정하고 멤버 변수 앞에는 member를 의미하는 'm_' 접두어를 붙였습니다.

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

그리고 아래와 같이 CopyDynamicString, CleanUp 함수를 MyString 클래스에 추가합니다. CleanUp 함수는 FreeDynamicString 함수의 이름을 변경한 것이고 클래스 내부에 다양한 자료를 관리할텐데, 함수 이름이 문자열을 해제한다고 되어 있으면 나중에 함수 이름이 문제가 될 수 있을 것 같아서 정리한다는 뜻의 이름으로 변경한 것입니다.

그리고 추가할 때 각 함수로 전달되던 MyString *ap_my_str 형식의 매개 변수는 이제 this 포인터로 대체되기 때문에 매개 변수에 적은 필요가 없고, this-> 표현도 모두 생략 가능하기 때문에 아래와 같이 코드가 훨씬더 간단해 집니다. 그리고 CopyDynamicString 함수에는 메모리를 할당하기 전에 이미 할당된 메모리가 있으면 정리하고 다시 할당할 수 있게 CleanUp 함수를 먼저 호출하게 했습니다.

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

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

그리고 구조체를 사용할 때는 아래와 같이 변수를 초기화 했습니다. 하지만 이제는 m_len, mp_str 멤버 변수가 private 접근 지정자를 사용하기 때문에 아래와 같이 사용하면 오류가 발생합니다. 그리고 이렇게 코드를 사용하게 되면 멤버 변수가 추가 되거나 위치가 변경되거나 하면 아래의 코드도 변경해야 하는 불편함이 생갑니다.

```
MyString my_str = { 0, NULL };
```

그래서 변화에 잘 대처할 수 있도록 아래와 같이 클래스에 Init 함수를 추가해서 객체를 초기화하는 역할로 사용합니다.

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

그리고 멤버 변수의 접근 지정자 때문에 오류가 나는 코드가 하나 더 있습니다. 아래의 코드이고 private 접근 지정자 때문에 mp_str과 m_len 변수의 값을 반환하는 함수를 클래스에 추가해서 코드를 수정해도 됩니다.

```
if (my_str.p_str != NULL) printf("%s : %d\n", my_str.p_str, my_str.len);
```

만약, 클래스에 문자열의 주소를 얻는 GetStringAddress 함수, 문자열의 길이를 얻는 GetStringLength 함수를 추가했다면 아래와 같은 코드로 수정될 것입니다. 아마도 이 코드를 작성하는 개발자가 불편하다거나 어렵다는 이야기가 나올만한 코

드가 될 것입니다.

```
if (my_str.GetStringAddress() != NULL)
    printf("%s : %d\n", my_str.GetStringAddress(), my_str.GetStringLength(()));
```

그래서 이런 방법 보다는 클래스에 아래와 같이 문자열을 출력하는 멤버 함수를 추가해서 해결하는 것이 더 좋습니다.

```
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;
        }
    }
};
```

이렇게 하면 초보 개발자에게 제공할 MyString 클래스가 완성되어 있습니다. C 언어 코드와 비교했을 때 코드가 더 길어지긴 했지만 포인터를 사용하는 코드가 많이 사라지고 단순해져서 코드를 이해하기는 더 쉬워졌을 것입니다. 물론 기능이나 데이터 사용 조건에 변화가 생기더라도 기존 코드보다는 클래스를 사용한 코드가 변화에 더 잘 대처할 것입니다.

이제 MyString 클래스를 사용하도록 main 함수를 수정해보면 다음과 같습니다.

```
#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;
}
```

클래스 코드 때문에 코드가 복잡해지긴 했지만 main 함수에서 MyString 클래스를 사용해서 작업하는 개발자 입장에서는 함수로 제공되는 코드보다 객체로 제공되는 코드가 사용하기도 쉽고 이해하기도 더 쉬울 것입니다.

댓글 등록순 최신순

조민희 3
객체 지향의 맛을 조금이나마 느낄 수 있는 강좌인 것 같습니다 ! 유익한 강좌 감사드립니다 :)
2022.07.06. 23:52 답글쓰기

김성엽 M 작성자
늦은시간까지 공부하느라 고생이 많군요 ㅎㅎ 파이팅입니다!

수고하셨습니다
2022.07.06. 23:55 답글쓰기

해피파파 3
클래스안으로 함수를 넣음으로서 포인터화살표를 없앴다는 것이 핵심이군요. c++ --> c 로 만들려면, 함수를 꺼내고, 화살표를 살리면 되겠네요?
2022.11.09. 01:15 답글쓰기

김성엽 M 작성자
ㅎㅎ 네~ this 포인터 대신에 매개 변수로 객체(구조체) 포인터를 매개 변수로 추가하고 함수 호출할 때 해당 객체(구조체)의 주소를 명시적으로 넘겨주는 코드를 사용하면 됩니다.
2022.11.09. 01:29 답글쓰기

황금잉어가물치 3
FreeDynamicString함수의 *ap_str 매개변수에 대해 질문드립니다. 해당 *ap_str 매개변수 main함수에서 *p_str값을 전달 받는데 CopyDynamics에서 이중 포인트로 전달받아 memcpy_s를 이용해 행렬로 입력된 값을 주소로 복사되었습니다. 그럼 해당 *p_str를 해제하기 위해 FreeDynamicString으로 전달해주려면 마찬가지로 이중 포인터로 보내줘서 해제하고 NULL값을 넣어야 하는게 아닌지요? 물론 행렬로 복사되어 FreeDynamicString함수로 동일한 주소값이 잘 전달되겠지만 이중 포인터로 전달하는게 맞는게 아닌지 궁금합니다.

2022.11.12. 18:08 답글쓰기

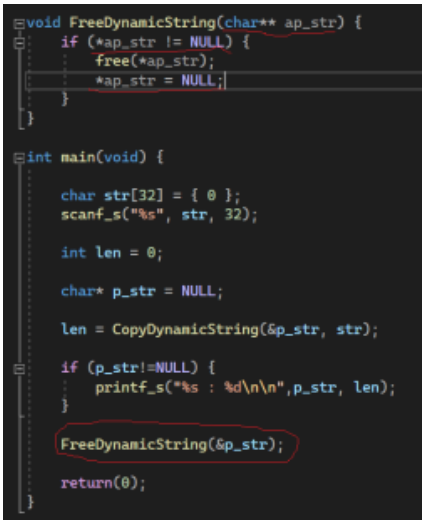
김성엽 M 작성자

메모리를 해제했다고 꼭 NULL로 다시 초기화할 필요는 없습니다. 보통 선택적인 경우가 많아서 이렇게 한거에요 ㅎ

2022.11.12. 18:14 답글쓰기

황금잉어가물치 3

위 질문에 대한 수정사항인데 이렇게 해야하는게 맞니 않는지요?



```
void FreeDynamicString(char** ap_str) {
    if (*ap_str != NULL) {
        free(*ap_str);
        *ap_str = NULL;
    }
}

int main(void) {

    char str[32] = { 0 };
    scanf_s("%s", str, 32);

    int len = 0;

    char* p_str = NULL;

    len = CopyDynamicString(&p_str, str);

    if (p_str!=NULL) {
        printf_s("%s : %d\n\n",p_str, len);
    }

    FreeDynamicString(&p_str);

    return(0);
}
```

2022.11.12. 18:05 답글쓰기

김성엽 M 작성자

NULL 꼭 넣어야 한다면 이렇게 하셔도 됩니다~ :)

2022.11.12. 18:13 답글쓰기

황금잉어가물치 3

김성엽 감사합니다.

2022.11.12. 18:31 답글쓰기

티모 3

함수를 만드는 사람이 되어야 ...

2022.11.14. 23:38 답글쓰기

카일 3

주옥같은 강의 잘 들었습니다. 감사합니다~^^

2022.11.17. 06:37 답글쓰기

dh221009

댓글을 남겨보세요

등록