# 컴퓨터공학 All in One

C/C++ 문법, 자료구조 및 심화 프로젝트 (나동빈)
제 53강 - C++의 스마트 포인터

# 학습 목표

C++의 스마트 포인터

1) C++의 스마트 포인터의 원리를 이해하고 이를 C++로 구현할 수 있습니다.

# C++의 스마트 포인터

스마트 포인터

C++의 스마트 포인터(Smart Pointer)는 프로그래머의 실수로 메모리 누수(Memory Leak)을 방어하기 위한 수단으로, 포인터처럼 동작하는 클래스 템플릿(Class Template)입니다.

기본적으로 힙 영역에 동적 할당된 메모리를 해제하기 위해서는 delete 키워드를 쓰면 됩니다. 스마트 포인터를 이용하면 메모리 누수를 더 효과적으로 방지할 수 있기 때문에 컴퓨터 시스템의 안정성을 높일 수 있습니다.

# C++의 스마트 포인터

스마트 포인터

일반적으로 new 키워드를 이용해서 기본 포인터가 특정한 메모리 주소를 가리키도록 초기화 한 이후에 스마트 포인터에 해당 포인터를 넣어서 사용할 수 있습니다.

이렇게 정의된 스마트 포인터는 수명을 다했을 때 소멸자가 delete 키워드를 이용해 할당된 메모리들을 자동으로 해제하는 기능을 수행합니다.

# C++의 스마트 포인터

스마트 포인터

- unique_ptr: 하나의 스마트 포인터가 특정한 객체를 처리할 수 있도록 합니다.
- shared_ptr: 특정한 객체를 참조하는 스마트 포인터가 총 몇 개인지를 참조합니다.
- weak_ptr: 하나 이상의 shared_ptr 인스턴스가 소유하는 객체에 대한 접근을 제공합니다.

# C++의 스마트 포인터

## unique_ptr

C++에서 하나의 스마트 포인터만이 특정한 객체를 처리하도록 할 때 unique_ptr를 사용할 수 있습니다.

이러한 스마트 포인터는 특정한 객체의 소유권을 가지고 있을 때만 소멸자가 객체를 삭제할 수 있습니다.

# C++의 스마트 포인터

## unique_ptr: 소유권 이전과 메모리 할당 해제

```
#include <iostream>

using namespace std;

int main(void) {
  unique_ptr<int> p1(new int(10));
  unique_ptr<int> p2;
  cout << "스마트 포인터 1: " << p1 << '\n';
  cout << "스마트 포인터 2: " << p2 << '\n';
  cout << "--- 소유권 이전 ---\n";
  p2 = move(p1); // 소유권 이전
  cout << "스마트 포인터 1: " << p1 << '\n';
  cout << "스마트 포인터 2: " << p2 << '\n';
  cout << "--- 메모리 할당 해제 ---\n";
  p2.reset(); // 메모리 할당 해제
  cout << "스마트 포인터 1: " << p1 << '\n';
  cout << "스마트 포인터 2: " << p2 << '\n';
  system("pause");
}
```

# C++의 스마트 포인터

## unique_ptr: 객체에 접근하기

```
#include <iostream>

using namespace std;

int main(void) {
  unique_ptr<int> p1(new int(10));
  cout << *p1 << '\n'; // 관리하고 있는 객체를 반환합니다.
  system("pause");
}
```

# C++의 스마트 포인터

## unique_ptr: 메모리 해제 이후에 객체 접근

```
#include <iostream>

using namespace std;

int main(void) {
  int* arr = new int[10];
  unique_ptr<int> p1(arr);
  for (int i = 0; i < 10; i++) {
    arr[i] = i;
  }
  for (int i = 0; i < 10; i++) {
    cout << arr[i] << ' ';
  }
  p1.reset();
  cout << '\n';
  for (int i = 0; i < 10; i++) {
    cout << arr[i] << ' ';
  }
  system("pause");
}
```

# C++의 스마트 포인터

## shared_ptr

C++에서 shared_ptr은 하나의 특정한 객체를 참조하는 스마트 포인터의 개수가 몇 개인지를 참조합니다.

특정한 객체를 새로운 스마트 포인터가 참조할 때마다 참조 횟수(Reference Count)가 1씩 증가하며, 각 스마트 포인터의 수명이 다할 때마다 1씩 감소합니다. 결과적으로 참조 횟수가 0이 되면 delete 키워드를 이용해 메모리에서 데이터를 자동으로 할당 해제합니다.

Fast campus

# C++의 스마트 포인터

## shared_ptr

```
#include <iostream>

using namespace std;

int main(void) {
  int* arr = new int[10];
  shared_ptr<int> p1(arr);
  cout << p1.use_count() << '\n';
  shared_ptr<int> p2(p1);
  cout << p1.use_count() << '\n';
  shared_ptr<int> p3 = p2;
  cout << p1.use_count() << '\n';
  p1.reset();
  p2.reset();
  p3.reset();
  cout << p1.use_count() << '\n';
  system("pause");
}
```

# C++의 스마트 포인터

## weak_ptr

C++에서 weak_ptr은 하나 이상의 shared_ptr 객체가 참조하고 있는 객체에 접근할 수 있습니다. 하지만 해당 객체의 소유자의 수에는 포함되지 않는 스마트 포인터입니다.

일반적으로 서로가 상대방을 가리키는 두 개의 shared_ptr이 있다면, 참조 횟수는 0이 될 수 없기 때문에 메모리에서 해제될 수 없습니다. weak_ptr은 이러한 순환 참조(Circular Reference) 현상을 제거하기 위한 목적으로 사용할 수 있습니다.

# C++의 스마트 포인터

## weak_ptr

```cpp
#include <iostream>

using namespace std;

int main(void) {
  int* arr = new int(1);
  shared_ptr<int> sp1(arr);
  weak_ptr<int> wp = sp1; // wp는 참조 횟수 계산에서 제외합니다.
  cout << sp1.use_count() << '\n'; // 1로 동일합니다.
  cout << wp.use_count() << '\n';
  if (true) {
    shared_ptr<int> sp2 = wp.lock(); // shared_ptr 포인터 반환
    cout << sp1.use_count() << '\n';
    cout << wp.use_count() << '\n';
  }
  // 스코프(Scope)를 벗어나므로 sp2가 해제됩니다.
  cout << sp1.use_count() << '\n';
  cout << wp.use_count() << '\n';
  system("pause");
}
```

Fast campus

# 배운 내용 정리하기

C++의 스마트 포인터

1) C++에서는 스마트 포인터(Smart Pointer)를 이용해 메모리 자원을 효과적으로 관리할 수 있습니다.