const와 static

날짜	@2023년 3월 13일

Const

변수를 **상수화**하는 목적으로 사용된다. const 선언은 변수 뿐 아니라 *포인터 변수, 함수, 객체*에도 사용이 가능하다.

변수 & 포인터 변수

1. ***const*** **int num = 10;**

   ⇒ 변수 num을 상수화 한다. 따라서 **num의 값**을 변경할 수 없다.

2. ***const*** **int * ptr1 = &val1;**

   ⇒ 포인터 ptr1을 이용해 **val1의 값**을 변경할 수 없다.

   (단, val1 = 20; 처럼 ptr1을 사용하지 않는 다른 방법으로는 수정이 가능함)

3. **int * *const* ptr2 = &val2;**

   ⇒ 포인터 ptr2를 상수화 한다. 따라서 **ptr2의 값**을 변경할 수 없다.

(단, *ptr2 = 100; 과 같이 ptr2가 가리키는 대상에 저장된 값을 변경하는 연산은 가능함)

4. ***const*** **int * *const* ptr3 = &val3;**

   ⇒ 포인터 ptr3과 val3을 상수화 한다. 따라서 **ptr3의 값**을 변경할 수 없으며 ptr3을 이용해 **val3의 값**을 변경할 수 없다.

객체

1. ***const*** **SoSimple sim(20)**;

   ⇒ 이 객체의 데이터 변경을 허용하지 않는다. 따라서 이 객체를 대상으로는 **const 멤버 함수만** 호출이 가능하다.

함수

1. **int GetX() *const*;**
2. **int GetY() *const*;**
3. **void ShowRecInfo() *const*;**

   ⇒ 이 함수 내에서는 멤버 변수에 저장된 값을 변경할 수 없다. 우회해서 값을 변경하는 상황을 방지하기 위해 const 함수 내부에서는 const가 아닌 함수를 호출할 수 없다.

const_cast

const_cast<T>(expr)

const의 특성을 제거할 때 사용한다.

아래 코드에서 const char * name은 원래 ShowString의 매개변수로 전달될 수 없다. 하지만 const_cast로 일시적으로 const의 특성을 제거하면서 허용된다.

```
void ShowString(char* str)
{
    cout<<str<<endl;
}

int main()
{
    const char * name = "Kim Seon Hyo"
```

```
    ShowString(const_cast<char*>(name));

    return 0;
}
```

문제

```
void ShowData(const int * ptr)
{
    int * rptr = ptr;
    printf("%d \n", *rptr);
    *rptr = 20;
}

int main()
{
    int num = 10;
    int * ptr = &num;
    ShowData(ptr);
    return 0;
}
```

1. 위 코드의 문제점을 말하시오.
    a. ShowData 함수의 매개변수인 ptr은 const 선언으로 상수화 되었다. 따라서 ptr을 사용해 값을 변경할 수 없다. 그러나 ShowData 함수 내부에서 ptr을 rptr에 대입하고 있기 때문에 ptr에 const 선언을 추가한 것이 의미 없는 상황이 되어 버렸다.

const 함수 내부에서 값을 변경할 수 있는 방법? static을 사용한다… 였던 거 같은데 그리고 const_cast 사용하기?

Static

클래스의 static 멤버변수

static으로 선언된 멤버변수는 객체의 멤버가 아니다.

1. **특징**

a. 클래스의 멤버변수를 static으로 선언하는 경우 ***해당 변수에 한해서 생성되는 모든 객체들이 값을 공유***한다. static 멤버변수는 전역변수와 동일하게 취급되지만 전역변수와는 다르게 접근 권한이 제한되어 있다. *(즉, static 멤버변수는 클래스 외부에 위치한다.)*

b. public으로 선언되면 클래스의 이름을 이용해서 호출이 가능하다. (ex. `SoSimple::simObjCnt`)

2. **초기화**

a. 변수를 공유하는 특성을 유지하기 위해 static 멤버변수는 클래스 외부에서 초기화 한다.

b. **const static** 을 사용하면 선언과 동시에 초기화가 가능하다.

```
class SoSimple
{
private:
    static int simStatic = 100;
    static int simObjCnt;
public:
    SoSimple()
    {
        simObjCnt++;
        cout<<simObjCnt<<"번째 SoSimple 객체"<<endl;
    }
};
int SoSimple::simObjCnt = 0;

int main()
{
    SoSimple sim1;
    SoSimple sim2;
    SoSimple sim3;
}
```

```
1번째 SoSimple 객체
2번째 SoSimple 객체
3번째 SoSimple 객체
```

클래스의 static 멤버함수

static으로 선언된 멤버함수는 객체의 멤버가 아니다.

따라서 static 멤버함수 내부에서는 ***static으로 선언된 멤버변수 혹은 멤버변수***에만 접근할 수 있다.

```
class SoSimple
{
private:
    int num1;
    static int num2;
public:
    SoSimple(int n):num1(n)
    {}
    static void Adder(int n)
    {
        num1+=n; // 컴파일 에러 발생
        num2+=n;
    }
};
int SoSimple::num2 = 0;
```

static 지역변수

지역변수에 static이 선언되는 경우 해당 변수는 전역변수와 동일한 시기에 할당되고 소멸된다. 저장되는 메모리 공간 또한 전역변수와 동일하다. 그러나 전역변수와는 달리 ***선언된 함수 내부에서만 접근***이 가능하다.

따라서 아래 코드처럼 **함수의 실행이 종료된 이후에도 값을 저장**할 수 있다.

```
void SimpleFunc()
{
    static int num1 = 0;
    int num2 = 0;
    num1++, num2++;
    printf("static: %d, local: %d \n", num1, num2);
}

int main()
{
    for(int i=0; i<3; i++)
        SimpleFunc();
    return;
}
```

```
static: 1, local: 1
static: 2, local: 1
static: 3, local: 1
```

static 전역변수

외부 파일에서의 접근을 허용하지 않을 때 사용한다.

static 함수

static 전역변수처럼 동일한 파일 내에서만 함수 접근이 가능하다.

static_cast

static_cast<T>(expr)

*상속관계에 있는 클래스의 포인터 및 참조형 데이터*의 형 변환 혹은 *기본 자료형 데이터*의 형 변환에서 사용된다.