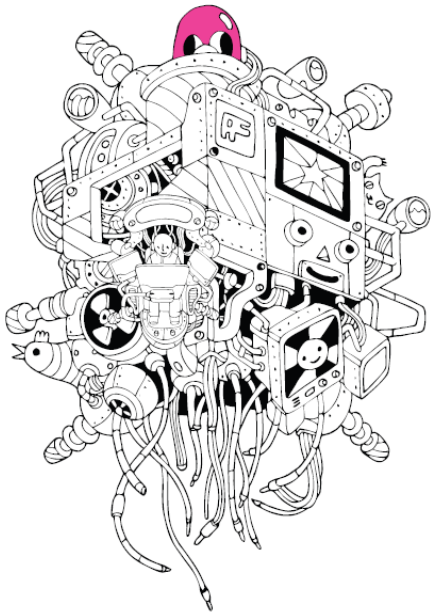
# 윤성우의
# 열혈 C++ 프로그래밍



윤성우 저 열혈강의 C++ 프로그래밍 개정판

## Chapter 16. C++ 형 변환 연산자와 맺는 글

# 윤성우의

# 열혈 C++ 프로그래밍

## Chapter 16-1. C++에서의 형 변환 연산

윤성우 저 열혈강의 C++ 프로그래밍 개정판

# 모기를 잡으려면 모기약을 써야지~



√ 아래에서 보이는 유형의 논란과 문제점 때문에 C++에서는 총 4개의 형 변환 관련 연산자를 제공하고 있다.

static_cast, const_cast, dynamic_cast, reinterpret_cast

```
class Car
{
private:
    int fuelGauge;
public:
    Car(int fuel) : fuelGauge(fuel)
    { }
    void ShowCarState() { cout<<"잔여 연료량: "<<fuelGauge<<endl; }
};

class Truck : public Car
{
private:
    int freightWeight;
public:
    Truck(int fuel, int weight)
        : Car(fuel), freightWeight(weight)
    { }
    void ShowTruckState()
    {
        ShowCarState();
        cout<<"화물의 무게: "<<freightWeight<<endl;
    }
};
```

```
int main(void)
{
    Car * pcar1=new Truck(80, 200);
    Truck * ptruck1=(Truck *)pcar1;
    ptruck1->ShowTruckState();
    cout<<endl;
    Car * pcar2=new Car(120);
    Truck * ptruck2=(Truck *)pcar2;
    ptruck2->ShowTruckState();
    return 0;
}
```

문제는 없으나 의도한 바인지 아닌지 알 수 없는 코드

프로그래머의 실수가 분명함! 그러나 컴파일러는 에러를 일으키지 않는다.

# dynamic_cast: 상속관계에서의 안전한 형 변환



```
class Car
{
    // 예제 PowerfullCasting.cpp의 Car 클래스와 동일
};

class Truck : public Car
{
    // 예제 PowerfullCasting.cpp의 Truck 클래스와 동일
};

int main(void)
{
    Car * pcar1=new Truck(80, 200);
    Truck * ptruck1=dynamic_cast<Truck*>(pcar1);   // 컴파일 에러

    Car * pcar2=new Car(120);
    Truck * ptruck2=dynamic_cast<Truck*>(pcar2);   // 컴파일 에러

    Truck * ptruck3=new Truck(70, 150);
    Car * pcar3=dynamic_cast<Car*>(ptruck3);   // 컴파일 OK!
    return 0;
}
```

의도한 바 일수 있다. 그리고 이러한 경우에는 staic_cast 형 변환 연산자를 사용해야 한다.

## dynamic_cast<T>(expr)

**포인터 또는 참조자인 expr을 T 형으로 변환하되 안전한 형 변환만 허용을 한다.**

**여기서 말하는 안전한 형 변환이란, 유도 클래스의 포인터 및 참조자를 기초 클래스의 포인터 및 참조자로 형 변환하는 것을 의미한다.**

# static_cast: A 타입에서 B 타입으로



```
class Car
{
    // 예제 PowerfullCasting.cpp의 Car 클래스와 동일
};

class Truck : public Car
{
    // 예제 PowerfullCasting.cpp의 Truck 클래스와 동일
};

int main(void)
{
    Car * pcar1=new Truck(80, 200);
    Truck * ptruck1=static_cast<Truck*>(pcar1);    // 컴파일 OK!
    ptruck1->ShowTruckState();
    cout<<endl;

    Car * pcar2=new Car(120);
    Truck * ptruck2=static_cast<Truck*>(pcar2);    // 컴파일 OK! 그러나!
    ptruck2->ShowTruckState();
    return 0;
}
```

## static_cast<T>(expr)

**포인터 또는 참조자인 expr을 무조건 T형으로 변환하여 준다.**

**단! 형 변환에 따른 책임은 프로그래머가 져야 한다.**

**static_cast 연산자는 dynamic_cast 연산자와 달리, 보다 많은 형 변환을 허용한다. 하지만 그에 따른 책임도 프로그래머가 져야 하기 때문에 신중히 선택해야 한다.**

**dynamic_cast 연산자를 사용할 수 있는 경우에는 dynamic_cast 연산자를 사용해서 안전성을 높여야 하며, 그 이외의 경우에는 정말 책임질 수 있는 상황에서만 제한적으로 static_cast 연산자를 사용해야 합니다**

# static_cast: 기본 자료형 간 변환



```
int main(void)
{
    int num1=20, num2=3;
    double result=20/3;
    cout<<result<<endl;
    . . . .
}
```

**C 스타일 형 변환**

```
double result=(double)20/3;
double result=double(20)/3;
```

**C++ 스타일 형 변환**

```
double result=static_cast<double>(20)/3;
```

**static_cast는 기본 자료형간 형 변환도 허용한다.**

```
int main(void)
{
    const int num=20;
    int * ptr=(int*)&num;
    *ptr=30;
    cout<<*ptr<<endl;
    float * adr=(float*)ptr;
    cout<<*adr<<endl;
    . . . .
}
```

const 제거!

상속과 관계 없는 포인터 형으로의 형 변환

**static_cast 연산자는 '기본 자료형 간의 형 변환'과**
**'클래스의 상속관계에서의 형 변환'만 허용!**

**C언어의 형 변환 연산자는 왼편에서와 같은 경우에도(모든 경우에) 형 변환을 허용.**

**따라서 제한적으로 허용하는 static_cast 연산자가 훨씬 안정적이다.**

# const_cast: const의 성향을 제거하라!



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

void ShowAddResult(int& n1, int& n2)
{
    cout<<n1+n2<<endl;
}

int main(void)
{
    const char * name="Lee Sung Ju";
    ShowString(const_cast<char*>(name));

    const int& num1=100;
    const int& num2=200;
    ShowAddResult(const_cast<int&>(num1), const_cast<int&>(num2));
    return 0;
}
```

## const_cast<T>(expr)

**expr에서 const의 성향을 제거한 T형 데이터로 형 변환하라!**

# reinterpret_cast: 상관없는 자료형으로의 형 변환

```
class SimpleCar { . . . . };
class BestFriend { . . . . };
```

서로 아무런 관련이 없는 두 클래스

reinterpret_cast<T>(expr)

expr을 T 형으로 형 변환하는데 expr의 자료형과 T는 아무런 상관관계를 갖지 않는다.

```
int main(void)
{
    SimpleCar * car=new Car;
    BestFriend * fren=reinterpret_cast<BestFriend*>(car);
    . . . .
}
```

형 변환의 결과는 예측하지 못한다.

```
int main(void)
{
    int num=0x010203;
    char * ptr=reinterpret_cast<char*>(&num);

    for(int i=0; i<sizeof(num); i++)
        cout<< static_cast<int>(*(ptr+i)) <<endl;

    return 0;
}
```

바이트 별 정수의 크기 출력하기

reinterpret_cast 형 변환 연산자의 적절한 사용의 예

```
3
2
1
0
```

실행결과

# dynamic_cast 두 번째 이야기: Polymorphic 클래스

**형 변환 연산의 기본규칙**

- 상속관계에 놓여있는 두 클래스 사이에서, 유도 클래스의 포인터 및 참조형 데이터를 기초 클래스의 포인터 및 참조형 데이터로 형 변환할 경우에는 dynamic_cast 연산자를 사용한다.
- 반대로, 상속관계에 놓여있는 두 클래스 사이에서, 기초 클래스의 포인터 및 참조형 데이터를 유도 클래스의 포인터 및 참조형 데이터로 형 변환할 경우에는 static_cast 연산자를 사용한다.

```
class SoSimple      // Polymorphic 클래스!
{                   // ShowSimpleInfo가 가상함수이므로
public:
    virtual void ShowSimpleInfo()
    {
        cout<<"SoSimple Base Class"<<endl;
    }
};

class SoComplex : public SoSimple
{
public:
    void ShowSimpleInfo()  // 이것 역시 가상함수!
    {
        cout<<"SoComplex Derived Class"<<endl;
    }
};
```

**아래의 예에서 보이듯이 기초 클래스가 Polymorphic 클래스라면 유도 클래스로의 포인터 및 참조형으로의 형 변환은 허용이 된다!**

```
int main(void)
{
    SoSimple * simPtr=new SoComplex;
    SoComplex * comPtr=dynamic_cast<SoComplex*>(simPtr);
    comPtr->ShowSimpleInfo();
    return 0;
}
```

# dynamic_cast와 static_cast의 차이



기초 클래스가 Polymorphic 클래스라면 유도 클래스의 포인터 및 참조형으로의 형 변환에는 dynamic_cast 연산자와 static_cast 연산자 모두 사용할 수 있다.
하지만 여전히 dynamic_cast 연산자는 안전성을 보장한다. 반면 static_cast 연산자는 안전성을 보장하지 않는다.

```
class SoSimple      // Polymorphic 클래스!
{                   // ShowSimpleInfo가 가상함수이므로
public:
    virtual void ShowSimpleInfo()
    {
        cout<<"SoSimple Base Class"<<endl;
    }
};

class SoComplex : public SoSimple
{
public:
    void ShowSimpleInfo()  // 이것 역시 가상함수!
    {
        cout<<"SoComplex Derived Class"<<endl;
    }
};
```

```
int main(void)
{
    SoSimple * simPtr=new SoComplex;
    SoComplex * comPtr=dynamic_cast<SoComplex*>(simPtr);
    . . . .
}
```

형 변환 OK!

```
int main(void)
{
    SoSimple * simPtr=new SoSimple;
    SoComplex * comPtr=dynamic_cast<SoComplex*>(simPtr);
    . . . .
}
```

형 변환 실패! NULL 반환

dynamic_cast 연산자는 위의 형 변환을 허용하지 않는다.
반면 static_cast 연산자는 허용을 한다. 물론 그 결과는 보장받지 못한다.

# bad_cast 예외



```
class SoSimple
{
public:
    virtual void ShowSimpleInfo()
    {
        cout<<"SoSimple Base Class"<<endl;
    }
};
class SoComplex : public SoSimple
{
public:
    void ShowSimpleInfo()
    {
        cout<<"SoComplex Derived Class"<<endl;
    }
};
```

```
int main(void)
{
    SoSimple simObj;
    SoSimple& ref=simObj;
    try
    {
        SoComplex& comRef=dynamic_cast<SoComplex&>(ref);
        comRef.ShowSimpleInfo();
    }
    catch(bad_cast expt)
    {
        cout<<expt.what()<<endl;
    }
    return 0;
}
```

```
Bad dynamic_cast!
```

실행결과

참조자 ref가 실제 참조하는 대상이 SoSimple 객체이기 때문에 SoComplex 참조형으로의 형 변환은 안전하지 못하다.

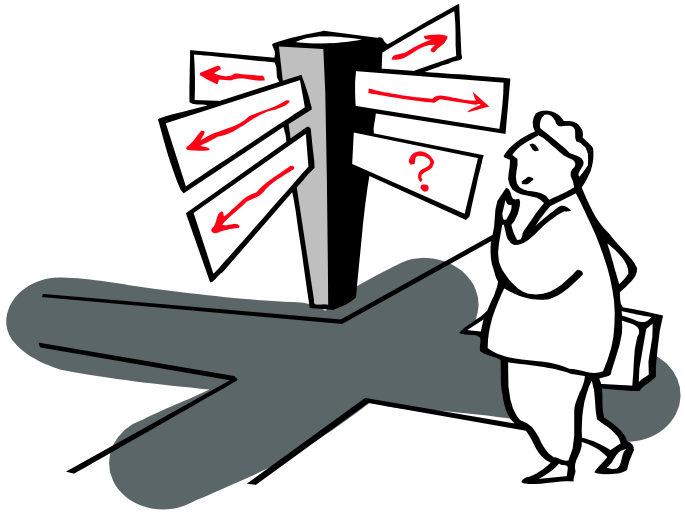
그리고 참조자를 대상으로는 NULL을 반환할 수 없기 때문에 이러한 상황에서는 bad_cast 예외가 발생한다.

# 윤성우의

# 열혈 C++ 프로그래밍

## Chapter 16-2. 맺는 글

윤성우 저 열혈강의 C++ 프로그래밍 개정판



Chapter 16이 끝났습니다.

그 동안 수고 많으셨습니다.