15장 기타 주제들

- auto_ptr
- 변환함수
- cast 연산자에 의한 명시적 형변환
- 실행시간 타입 정보 알아내기(RTTI)

1. auto_ptr

- 다음 프로그램의 문제점은 무엇인가?

```
void func(void)
{
    int *p = new int;
    cout << "양수 입력 : ";
    cin >> *p;

    if (*p <= 0) {
        cout << "양수를 입력해야 합니다" << endl;
        return;
    }

    cout << "입력 값 : " << *p << endl;
    delete p;
}

int main(void)
{
    for (int i = 0; i < 10; i++)
        func();

    return 0;
}
```

동적 할당 메모리를 해제하지 않고 리턴

int *p = new int;
변수 **p**가 사라지는 경우
동적으로 할당받은 메모리도
헤제할 수 있는 방법은 없을까???

1. auto_ptr

- 자동으로 해제 가능한 포인터 클래스 만들기

```
template <typename T>
class AutoPtr {
private :
    T *ptr;

public :
    AutoPtr(T *p) : ptr(p) { }          // 새로 할당한 메모리를 ptr에 대입
    T &operator*() { return (*ptr); }   // 역참조 연산자
    ~AutoPtr() { delete ptr; }          // 소멸자를 통해 delete 수행
};

int main(void)
{
    AutoPtr<int> p(new int);            // 동적 할당 및 AutoPtr 객체 생성
    *p = 5;
    cout << *p << endl;

    return 0;
}
```

내부적으로 포인터를 가지고 있음

함수가 끝날 때 → 지역변수 **p** 해제
→ 소멸자에 의해 동적 할당 메모리 해제!

1. auto_ptr

- 표준 C++ : auto_ptr 클래스 템플릿 제공
  - <memory> 헤더 파일 포함
  - 배열에 대한 동적 생성 및 해제 자동화 불가

```
#include <iostream>
#include <memory>
using namespace std;

int main(void)
{
    auto_ptr<int> p(new int);
    *p = 5;
    cout << *p << endl;

    return 0;
}
```

2. 변환함수

- 상속 관계에 있는 객체들 사이의 대입 관련 복습 (9.2절)
  - base 클래스 객체 = derived 클래스 객체 : 대입 가능
  - 그 외는 불가능
- 다음과 같은 대입이 가능하도록 만들려면 어떻게 해야 할까?

```
class base { ... };
class derived : public base { ... };
class another { ... };
base b; derived d; another a; int i;
b = d;      // ① O, base 객체 = derived 객체
d = b;      // ② X, derived 객체 = base 객체
a = b;      // ③ X, another 객체 = base 객체
b = i;      // ④ X, base 객체 = int 변수
i = b;      // ⑤ X, int 변수 = base 객체
```

1. 자동으로 가능한 경우는?
2. 대입 연산자 오버로딩으로 가능한 경우는?
3. 지금까지 배운 것만으로는 불가능한 경우는?

2. 변환함수

- 대입 연산자 오버로딩을 사용한 대입 해결 : ②, ③, ④

```
class base {
public :
    int x;

    base(int a = 0) { x = a; }        // ④의 해결, int → base 형변환
    void show(void) { cout << "base : " << x << endl; }
};

class derived : public base {
public :
    int y;

    derived(int a = 0, int b = 0) : base(a) { y = b; }
    void show(void) { base::show();
        cout << "derived : " << y << endl; }
    void operator=(const base &b) {  // ②의 해결, 대입 연산자 오버로딩
        x = b.x;
        y = b.x;
    }
};
```

int 값이 **base** 객체로 형변환 된 후 대입됨

2. 변환함수

- 대입 연산자 오버로딩을 사용한 대입 해결 : ②, ③, ④ (계속)

```
class another {
public :
    double z;

    another(double c = 0) { z = c; }
    void show(void) { cout << "another : " << z << endl; }
    void operator=(const base &b) {  // ③의 해결, 대입 연산자 오버로딩
        z = b.x;
    }
};

int main(void)
{
    base b; derived d; another a; int i = 1;
    b = d;          // ① O, base 객체 = derived 객체
    d = b;          // ② O, derived 객체 = base 객체
    a = b;          // ③ O, another 객체 = base 객체
    b = i;          // ④ O, base 객체 = int 변수
    //i = b;        // ⑤ X, int 변수 = base 객체

    return 0;
}
```

불가능?
해결 방법은? → 변환함수!

2. 변환함수

- 변환함수를 이용한 ⑤(int 변수 = base 객체)의 해결
  - 변환함수 문법 : operator int() { ...; return int값; }

```
class base {
public :
    int x;

    base(int a = 0) { x = a; }
    void show(void) { cout << "base : " << x << endl; }
    operator int();   //{ return (x * x); }     // ⑤의 해결, 변환함수
};

base::operator int() // 외부 정의
{
    return (x * x);
}
```

②, ③의 경우 대입 연산자 오버로딩을 사용하지 않고 변환함수로 해결 가능
→ 연습 문제 15.3

3. cast 연산자에 의한 명시적 형변환

- C 스타일의 명시적 형변환
  - (타입명) 변수명;
  - 형변환 예

```
class Base { };
class Derived : public Base { };
class Another { };
int i; int *p; double d; Base *b; Derived *d; Another *a;
i = (int) d;          // ① 가능, 묵시적 형변환 가능
p = (int *) &d;       // ② 가능
p = (int *) b;        // ③ 가능
b = (Base *) d;       // ④ 가능, 묵
d = (Derived *) b;    // ⑤ 가능
b = (Base *) a;       // ⑥ 가능
```

현재 b가 derived 객체를 가리킨다면? Ok!
base 객체를 가리킨다면? 논리적 오류 가능!

진짜 원한 것인가?

- C++ : C 스타일 사용 가능
  - 다양한 형변환을 구별하기 위한 4개의 명시적 형변환 연산자 제공
  - dynamic_cast, const_cast, static_cast, reinterpret_cast

3. cast 연산자에 의한 명시적 형변환

- dynamic_cast
  - 상속 관계인 클래스 객체 포인터 또는 참조에 대해 적용 가능
  - 가상 함수를 포함하고 있어야 함

VC++ 6.0의 경우 컴파일 옵션으로 /GR 추가

```
class Base {
public :
    int b;
    virtual void func() { cout << "Base" << endl; };
};

class Derived : public Base {
public :
    int d;
    void func() { cout << "Derived" << endl; }
};

int main(void)
{
    Base *b = new Derived(); //Base *b = new Base(); // 바꾸어 수행해 보라.
    Derived *d = dynamic_cast<Derived *> (b); // b가 가리키는 주소 대입

    if (d == NULL)
        cout << "형변환 실패" << endl;
    else
        cout << "형변환 성공" << endl;

    return 0;
}
```

b가 Derived 객체를 가리키고 있다면→성공
Base 객체를 가리키고 있다면→NULL

3. cast 연산자에 의한 명시적 형변환

- static_cast
  - 상속 관계인 클래스 객체 포인터 또는 참조에 대해 적용 가능
  - 앞의 예에서 Base *b = new Base(); 인 경우에도 형변환 성공
- const_cast
  - 포인터 변수값을 const형에서 일반형으로, 일반형에서 const형으로 변환
  - const를 제외하면 동일한 포인터 타입인 경우 형변환 가능

```
int main(void)
{
    const double *pi = new double(3.14);
    double *pi2 = const_cast<double *> (pi);
    *pi2 = 3.14159;

    cout << *pi << endl;
    cout << *pi2 << endl;

    return 0;
}
```

3. cast 연산자에 의한 명시적 형변환

- reinterpret_cast
  - 서로 무관한 타입 사이의 형변환 가능
    - int형과 포인터형
    - 무관한 클래스 포인터 사이의 형변환
  - Derived 클래스와는 무관한 Another 클래스가 있을 경우
    - Base *b = new Base();
    - Another *a = reinterpret_cast<Another *> (b); // 형변환 가능
- C++ 명시적 형변환 연산자의 역할
  - 안전한 형변환 제공
  - 가독성 증가 : 상황을 보다 쉽게 파악할 수 있음

4. 실행시간 타입 정보 알아내기 (RTTI)

- 다음과 같이 실행 도중에 base 포인터가 가리키고 있는 객체의 타입을 알 수 있을까?

```
void main(base *bp)
{
    if (bp가 가리키는 객체의 타입이 base이면)
        bp->func1();
    else (bp가 가리키는 객체의 타입이 derived이면) {
        derived *dp = (derived *) bp;
        dp->onlyDerivedFunc();
    }
}
```

  - derived 객체인 경우 derived 객체에만 있는 멤버 함수 호출을 원함

4. 실행시간 타입 정보 알아내기 (RTTI)

- dynamic_cast를 이용한 간접적 해결

```
class Base {
public :
    int b;
    virtual void func() { cout << "Base" << endl; };
};

class Derived : public Base {
public :
    int d;
    void func() { cout << "Derived" << endl; }
    void func2() { cout << "func2" << endl; }
};

int main(void)
{
    Base *b = new Derived();
    Derived *d = dynamic_cast<Derived *> (b); // 가능하다면 Derived로 변환

    if (d == NULL)    // Base 객체임
        cout << "형변환 실패" << endl;
    else              // Derived 객체임
        d->func2();

    return 0;
}
```

4. 실행시간 타입 정보 알아내기 (RTTI)

- RTTI(RunTime Type Information)
  - 변수, 포인터, 타입에 대한 정보를 알아낼 수 있는 방법 제공
  - typeid 연산자 → type_info 클래스 객체의 참조 반환

```
int a, b;
cout << typeid(a).name() << endl;      // 변수가 올 수도 있다.
cout << typeid(int).name() << endl;    // 타입이 올 수도 있다.
if (typeid(a) == typeid(b)) { ... }    // 두 변수의 타입이 같은지 비교할 수 있다.
if (typeid(a) == typeid(int)) { ... }  // 변수의 타입이 int인지 알아낼 수 있다.
```

```
int main(void)
{
    Base *b = new Derived();
    Derived *d;

    if (typeid(*b) == typeid(Derived)) {      // b 객체가 Derived 객체인지 검사
        d = (Derived *) b;
        cout << typeid(*d).name() << endl;
        d->func2();
    }
    else
        cout << "Derived 객체가 아닙니다" << endl;

    return 0;
}
```

<typeinfo> 헤더 파일 포함
VC++ 6.0의 경우 컴파일 옵션으로 **/GR** 추가